Development of Procedures and Tools for Calibration of I.C Engine Combustion Models

Undergraduate Honors Thesis

Presented in Partial Fulfillment of the Requirements for Graduation with Distinction at The Ohio State University

By

Christopher M. Hoops

* * * * *

The Ohio State University

2009

Defense Committee:

Professor Giorgio Rizzoni, Advisor

Professor Yann Guezennec

Approved by

Adviser

Undergraduate Program in Mechanical Engineering

ABSTRACT

As the world oil reserves continue to deplete, the need for more fuel efficient vehicles has become a necessity among the society to counteract the increasing fuel prices. In order to meet these demands advanced internal combustion engines are being developed; however the controls for these engines has become more complex, due to added degrees of freedom, which adds time to the calibration process. As a result, engine modeling has become popular for control design to reduce the calibration stages. This research developed a methodology for combustion analysis and modeling processes for advanced internal combustion engines. The methodology was valid over the entire operating range of the engine subject to changes in engine speed, manifold pressure, spark timing, intake cam position, and exhaust cam position. This research was part of a much larger effort to improve the modeling capabilities of advanced engines. The objective is to use “model based” techniques to facilitate these advanced engines entering the market, creating positive impacts on fuel economy and emissions. The overall project will consist of experimental set-up, data collection, analytical analysis and modeling.

ACKNOWLEDGMENTS

I would like to thank my Advisor Professor Rizzoni for allowing me the opportunity for an undergraduate research project and the assistance he has given me along the way. I would especially like to thank Dr. Marcello Canova who helped me with the burn rate analysis and guidance through the project and Dr. Shawn Midlam-Mohler with help in the experimental setup and data processing. I would also like to thank Ph.D. student Kenny Follen for his overall support in helping me through the project. Finally I would like to thank the Center of Automotive Research for allowing me to use the testing facility and equipment and General Motors for the financial support.

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Introduction

Over the past century the internal combustion (IC) engine has dominated the automotive industry. The IC engine has proven to be very reliable and robust and continues to dominate the automotive industry. However due to the recent fluctuations in oil prices and the societal emphasis on a "Greener Environment", heavy consumer demands have been placed on the automotive companies for vehicles with better efficiency, fuel economy and emissions without compromising performance. Developing higher fuel efficient engines will not only benefit the consumer at the pump, but also prolong the peak oil production and reduce emissions that are linked to climate change.

In order for the automotive companies to meet these consumer demands advanced IC engines are being developed with new technologies like variable valve timing (VVT) and gasoline direct injection (GDi). VVT allows the electronic control unit (ECU) to vary the valve timing over the engine operating range to achieve better efficiency, power and emission outputs. GDi injects fuel directly into the cylinder through a highly pressurized fuel rail and allows for lean burn operation in order to maximize efficiency

and reduce emissions. Without technologies like VVT and GDi the consumer demands and government regulations on emissions would be hard to meet.

However with these new technologies there are some associated drawbacks. The flexibility of these new technologies requires more advanced control strategies to be developed due to the added degrees of freedom to the engine. Added degrees of freedom allow for optimization of engine efficiency, fuel economy, emissions and power; however it also increases complexity in engine mapping. In the past all engine mapping has been completed through manual engine calibration, conducted in testing facilities. Using manual calibration, a significant amount of time is spent in the calibration stages, especially as the degrees of freedom are increased. Imagine mapping an advanced IC engine over the entire engine operating range trying to optimize spark timing, air fuel ratio, injection timing, intake valve opening, intake valve closing, intake valve lift, exhaust valve opening, exhaust valve closing, exhaust valve lift. As more technologies are introduced the degrees of freedom are increased and the mapping stages become more complicated.

In order to decrease the time engines spend in the calibration stages, control oriented engine models are replacing manual calibration. These engine models are developed and calibrated based on a relatively small amount of empirical data. Once calibrated these models are capable of being used for a significant portion of control design. The complete engine model can be broken down to a gas exchange model and combustion model. The focus of this project was on the combustion model side.

1.2 Literature Review

There has been extensive research in both fields that are associated with this research project. The first field deals with the calibration methods for control oriented combustion models and the second field deals with measurements needed for the calibration of control oriented combustion models. This literature review will cover the more prominent methods used for control oriented combustion modeling in both of these areas and be followed by the motivation for this project.

The most widely used method in calibrating control oriented combustion models is based on the Wiebe Function. This calibration method is discussed in the books (Ferguson and Kirkpatrick) and (Heywood), and has been used in many research applications. For example (Ponti, Serra and Siviero) uses the Wiebe function to model the combustion of a multi-jet diesel engine for up to four injections per cycle. A combination of Wiebe functions was used which was equal to the number of injections.

The Wiebe function is a measure of the burned mass fraction, or cumulative heat release, and is represented by the equation

$$x_b = 1 - \exp\left[-a\left(\frac{\theta - \theta_0}{\Delta\theta} \right)^{m+1} \right] \tag{1}$$

where θ is the crank angle, θ_0 is the start of combustion, $\Delta\theta$ is the total combustion duration ($x_b = 0$ to $x_b = 1$), and a and m are shape factors. The burned mass fraction is related to the heat release rate as a function of crank angle as,

$$\frac{dQ_{chem}}{d\theta} = Q_{in}\frac{dx_b}{d\theta} \tag{2}$$

where Q_{in} is the lower heating value of fuel times the mass of fuel present. This heat release rate can then be fit to the experimental heat release rate which is found using the experimental cylinder pressure. From the literature (Heywood), (Ferguson and Kirkpatrick) and (Dawson) the most common method to finding the experimental heat release for control oriented models uses a single zone model which assumes that the burned and unburned gases are uniformly mixed. This can be represented with the following equation,

$$\frac{Q_{chem}}{d\theta} = \frac{1}{\gamma - 1}V_{cyl}\frac{dp_{cyl}}{d\theta} + \frac{\gamma}{\gamma - 1}p_{cyl}\frac{dV_{cyl}}{d\theta} + \frac{Q_{heat}}{d\theta} \tag{3}$$

where p_{cyl} is the cylinder pressure V_{cyl} is the cylinder volume, Q_{heat} is the heat transfer through the cylinder walls and γ is the specific heat ratio. Functions are then determined for the Wiebe parameters based on the input control parameters to bridge the gaps between the Wiebe functions and complete the combustion model.

Special attention at been paid to the heat transfer through the cylinder walls and the specific heat ratio due to their affects on the heat release rate. To compensate for heat transfers lost through the cylinder wall the Woshni correlation (Heywood) has proven to be successful. The specific heat ratio is dependent on temperature, pressure, and composition of the mixture during combustion (Brunt and Pond). Iterative methods have been used to determine the composition within the cylinder using the burn mass fraction.

This process was used when modeling diesel combustion based on injection pulses in the research by (Ponti, Serra and Siviero).

From the literature it has been proven and emphasized that the most essential measurement in calibrating control oriented combustion models is the cylinder pressure (Heywood), (Ferguson and Kirkpatrick), (Yoon, Lee and Sunwoo), and (Ponti, Serra and Siviero). The Wiebe function previously discussed is an example of a combustion model based on the cylinder pressure. To measure the cylinder pressure most accurately piezoelectric pressure transducers are typically used due to their dynamic properties and good temperature stability (Heywood) and (Zhao and Ladommatos). However these sensors measure relative pressure therefore the cylinder pressure must be pegged to an absolute pressure in order to get true pressure. Since the combustion models are based on the cylinder pressure a large amount of research has been conducted to accurately peg the cylinder pressure.

Three distinct methods of pressure pegging where found in the literature (Zhao and Ladommatos), (Lee, Yoon and Sunwoo) and (Randolph). The first method pegged the cylinder pressure to the intake manifold pressure at the instant when the piston was at bottom dead center of the intake stroke. This method is proven to only be successful for unturned intake systems due to the noise created by a tuned system when the piston is at bottom dead center. The second method is to peg the cylinder pressure to the exhaust backpressure during the exhaust stroke. The pulsations in the exhaust manifold are minuscule and can be alleviated using an average over a discrete range of the exhaust stroke, thus making it a good choice for pressure pegging. The only downfall to this

method is that an absolute exhaust pressure sensor is needed to measure the backpressure and it has to withstand the high exhaust temperatures.

The final method research was the polytropic coefficient method. This method is based on the fact that engine compression follows the polytropic equation.

$$pV^{\alpha} = \text{constant} \tag{4}$$

Two methods were developed based on a fixed and variable polytropic coefficient (Lee, Yoon and Sunwoo). The variable polytropic coefficient method was found to be more accurate even though it was more susceptible to noise. This method was applied when developing a closed loop control method for start of combustion in a CRDI diesel engine using the cylinder pressure (Yoon, Lee and Sunwoo).

1.3 Motivation

From the literature, established and accepted models have been developed to process cylinder pressure data and to use cylinder pressure as the basis for control oriented modeling. However this research is usually focused on developing either a combustion model based on cylinder pressure or techniques for conditioning the cylinder pressure. There has been little effort in developing a systemic approach that brings these two fields together to decrease the amount of user interaction between collecting cylinder pressure data in the lab and developing a completed combustion model. The faster the models can be calibrated the faster they can be used for engine calibration thus saving time and money.

1.4 Project Objective

The goal of this project was to create a systematic methodology to construct a control oriented combustion model from raw cylinder pressure data collected in the laboratory. A set of tools were developed based on the methodology that combined the well established models from the literature and bridged any gaps needed to complete the entire process. These tools were automated to limit user interaction and decrease the time needed for building the combustion model. Once this model was completed it was implemented into a larger project that will model all the components and inputs to the engine. This complete model will be used in efforts to improve "model based" techniques to optimize fuel economy and engine efficiency for advanced engines in place of experimental techniques.

CHAPTER 2

EXPERIMENTAL DESCRIPTION

The experimentation was conducted at the Center for Automotive Research associated with the Ohio State University on a General Motors 2.4 L Eco-Tec advanced four-cylinder engine. The experimentation occurred in a test cell in which the engine was coupled to a 300 Hp AC dynamometer. The dynamometer was used to absorb the engine power and was able to operate in constant speed or torque mode using a dyne system controller. For all testing purposes the dynamometer was operated in constant speed mode, which meant the dynamometer kept the engine speed constant no matter the influence from the engine.

The Eco-Tec engine was equipped with variable valve timing (VVT), which used dual hydraulic cam phasors to change the valve timing. The system could independently shift the intake and exhaust cams over a range of 60 crank angle degrees (CAD) however the system could not vary the lift or duration of the cams. The minimum valve overlap, when both the intake and exhaust valves were open, of the system was 2 CAD and the maximum valve overlap was 122 CAD. The VVT system added more flexibility to the engine tuning process to achieve better performance, efficiency and emissions. Exact specifications of the engine can be seen in Appendix A.

2.1 Engine Instrumentation

The engine was instrumented with sensors to record and monitor temperatures, pressures, flows, emissions, speed and torque. The location of these sensors were strategically placed to ensure the quality and accuracy of the measurements since these measurements were used for the combustion model and real time monitoring to stay within testing constraints and engine limitations. Modifications to the engine and components were made as necessary to equip the engine with all the sensors, without compromising the functionality of the engine. A complete sensor list is displayed in Table 1 and Table 2. The highlighted measurements were needed to build the combustion model. All other measurements were used for monitoring. The numbers in the left hand column correspond to the numbers on Figure 1, Figure 2, and Figure 3 to show the location of each of the sensors. Figure 1 shows the top view of the engine coupled to the dynamometer and Figure 2 shows the engine from the side view. Figure 3 shows the facility intake system that brings fresh air to the engine intake system.

Table 1: Listing of all the temperatures and pressures acquired during testing, the highlighted items were relevant to the combustion model.

Temperatures		Descriptions	DAQ System
1	TC_1_0	Test Cell Temp	Low Frequency
2	TC_1_1	Before Catalyst Temp	Low Frequency
3	TC_1_2	After Catalyst Temp	Low Frequency
4	TC_1_3	Oil Sump / Drain Temp	Low Frequency
5	TC_1_4	Exhaust Runner 1 Temp	Low Frequency
6	TC_1_5	Exhaust Runner 2 Temp	Low Frequency
7	TC_1_6	Exhaust Runner 3 Temp	Low Frequency
8	TC_1_7	Exhaust Runner 4 Temp	Low Frequency
9	TC_1_9	Exhaust Manifold Temp	Low Frequency
10	TC_1_10	Coolant Out Temp	Low Frequency
11	TC_1_11	Coolant In Temp	Low Frequency
12	TC_1_12	High Bay Temp	Low Frequency
13	TC_1_13	Intake Runner 1 Temp	Low Frequency
14	TC_1_14	Intake Runner 2 Temp	Low Frequency
15	TC_1_15	Intake Runner 3 Temp	Low Frequency
16	TC_1_16	Intake Runner 4 Temp	Low Frequency
17	TC_1_17	Intake Runner 1 Wall	Low Frequency
18	TC_1_18	Intake Runner 2 Wall	Low Frequency
19	TC_1_19	Intake Runner 3 Wall	Low Frequency
20	TC_1_20	Intake Runner 4 Wall	Low Frequency
21	TC_1_21	Intake Manifold Temp	Low Frequency
22	TC_1_22	Inlet Horn Temp	Low Frequency
23	TC_1_23	Air Filter Temp	Low Frequency
24	TC_1_24	Before LFE Temp	Low Frequency
25	TC_1_25	After LFE Temp	Low Frequency
Pressures		**Descriptions**	
26	Cyl_1	In-Cylinder Pressure Cylinder 1	High Frequency
27	Cyl_2	In-Cylinder Pressure Cylinder 2	High Frequency
28	Cyl_3	In-Cylinder Pressure Cylinder 3	High Frequency
29	Cyl_4	In-Cylinder Pressure Cylinder 4	High Frequency
30	Int	Intake Manifold Pressure	High Frequency
31	Exh	Exhuast Manifold Pressure	High Frequency
32	P_Inlet_Horn	Intake Horn Pressure	Low Frequency
33	Barometric_Airbox	Intake Airbox Pressure	Low Frequency
34	Differnetial_P_LFE	Laminar Flow Element Diff. Pressure (MAF)	Low Frequency
35	Barometric_Pressure	Barometric Pressure	Low Frequency
36	Oil_Pressure	Oil Pressure in Crank Case	Low Frequency

Table 2: Listing of all the other measurements acquired during testing, the highlighted items were relevant to the combustion model.

Miscellaneous		Descriptions	
37	Crnk_sh_Enc	Crankshaft Encoder	High Frequency
38	Speed	Engine Speed - Dyno	Low Frequency
39	Torque	Engine Torque - Dyno	Low Frequency
40	UEGO_1	Oxygen Sensor (AFR)	Low Frequency
Emissions		**Descriptions**	
41	NO_x _1	Nitric Oxides Emiss.	Low Frequency
41	THC_1	Hydrocarbon Emiss.	Low Frequency
41	CO_2 _1	Carbon Dioxide Emiss.	Low Frequency
41	CO(H)_1	Carbon Monoxide Emiss. - high range	Low Frequency
41	CO(L)_1	Carbon Monoxide Emiss. - low range	Low Frequency
41	O_2 _1	Oxygen Emiss.	Low Frequency
41	H_2	Hydrogen Emiss.	Low Frequency
ECU Channels		**Descriptions**	
42	ETAS_1_1	Equivalence Ratio (used for AFR)	Low Frequency
42	ETAS_1_2	Engine Speed	Low Frequency
42	ETAS_1_3	Intake Manifold Pressure	Low Frequency
42	ETAS_1_4	Mass Air Flow through Intake (MAF)	Low Frequency
42	ETAS_1_5	Spark Timing	Low Frequency
42	ETAS_1_6	Intake Cam Timing	Low Frequency
42	ETAS_1_7	Exhuast Cam Timing	Low Frequency
42	ETAS_1_8	Engine Coolant Temerature	Low Frequency
42	ETAS_2_1	Throtlle Position	Low Frequency
42	ETAS_2_2	Throttle Area	Low Frequency
42	ETAS_2_3	Throttle Area_IND	Low Frequency
42	ETAS_2_4	NOT USED	Low Frequency
42	ETAS_2_5	NOT USED	Low Frequency
42	ETAS_2_6	NOT USED	Low Frequency
42	ETAS_2_7	NOT USED	Low Frequency
42	ETAS_2_8	NOT USED	Low Frequency

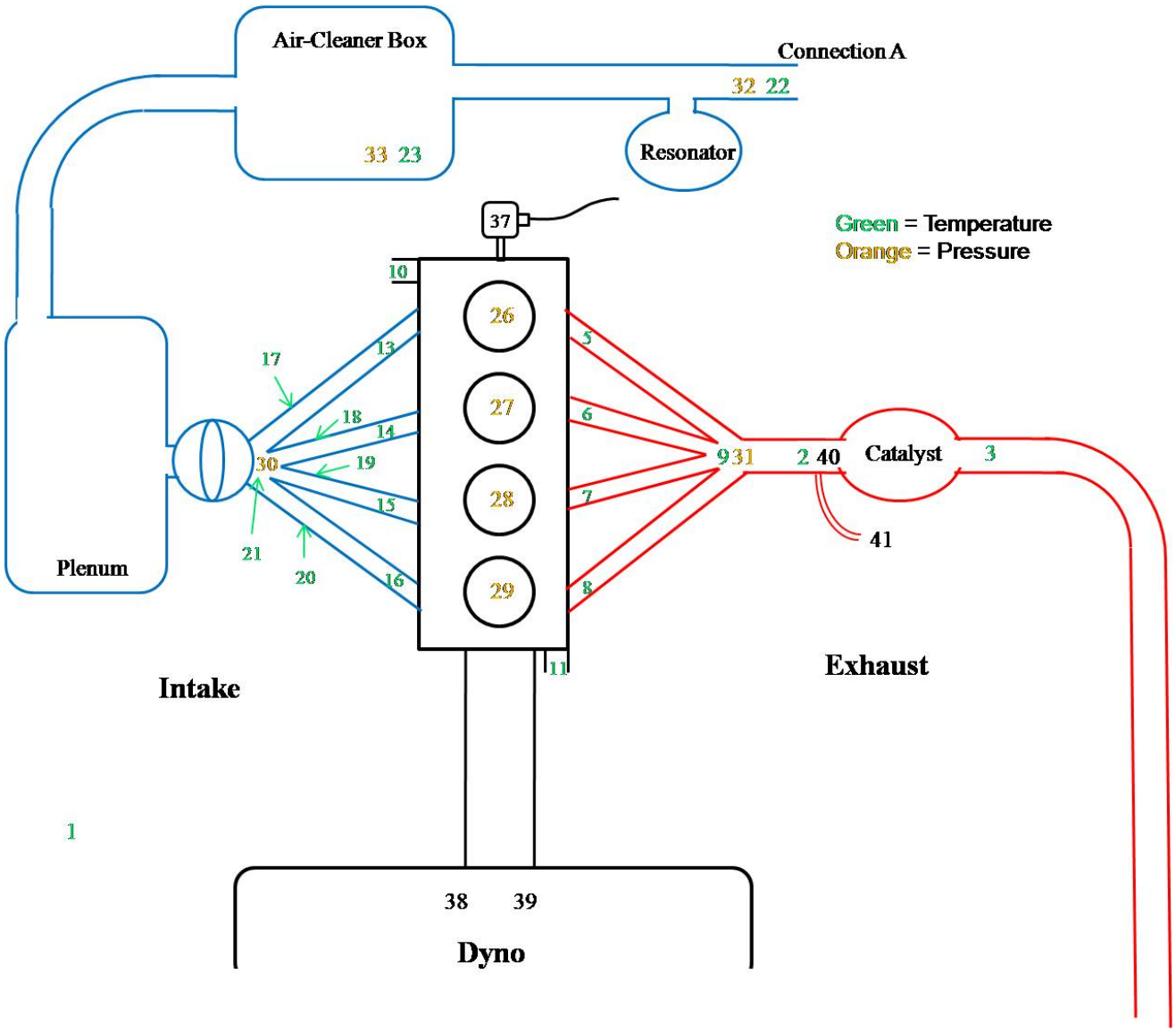


Figure 1: Top view of the engine, including the intake and exhaust system, and dynamometer coupling to show the location of the sensors.

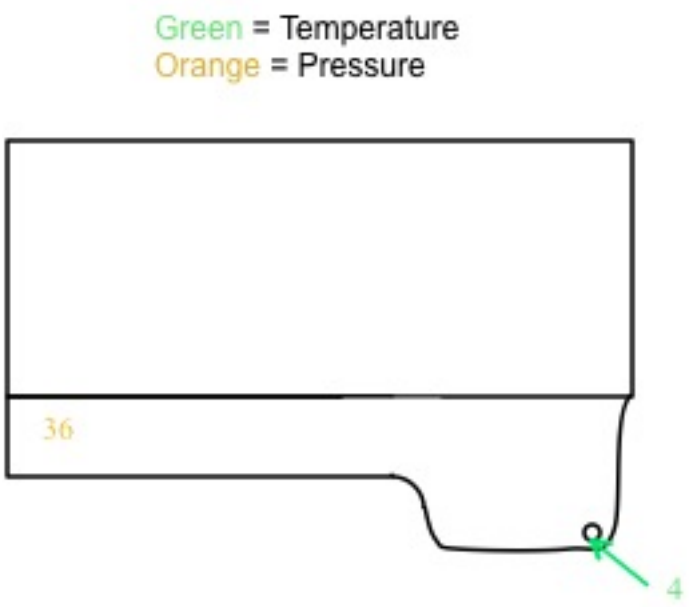


Figure 2: Side view of the engine to show the placement of the oil temperature and pressure sensors.

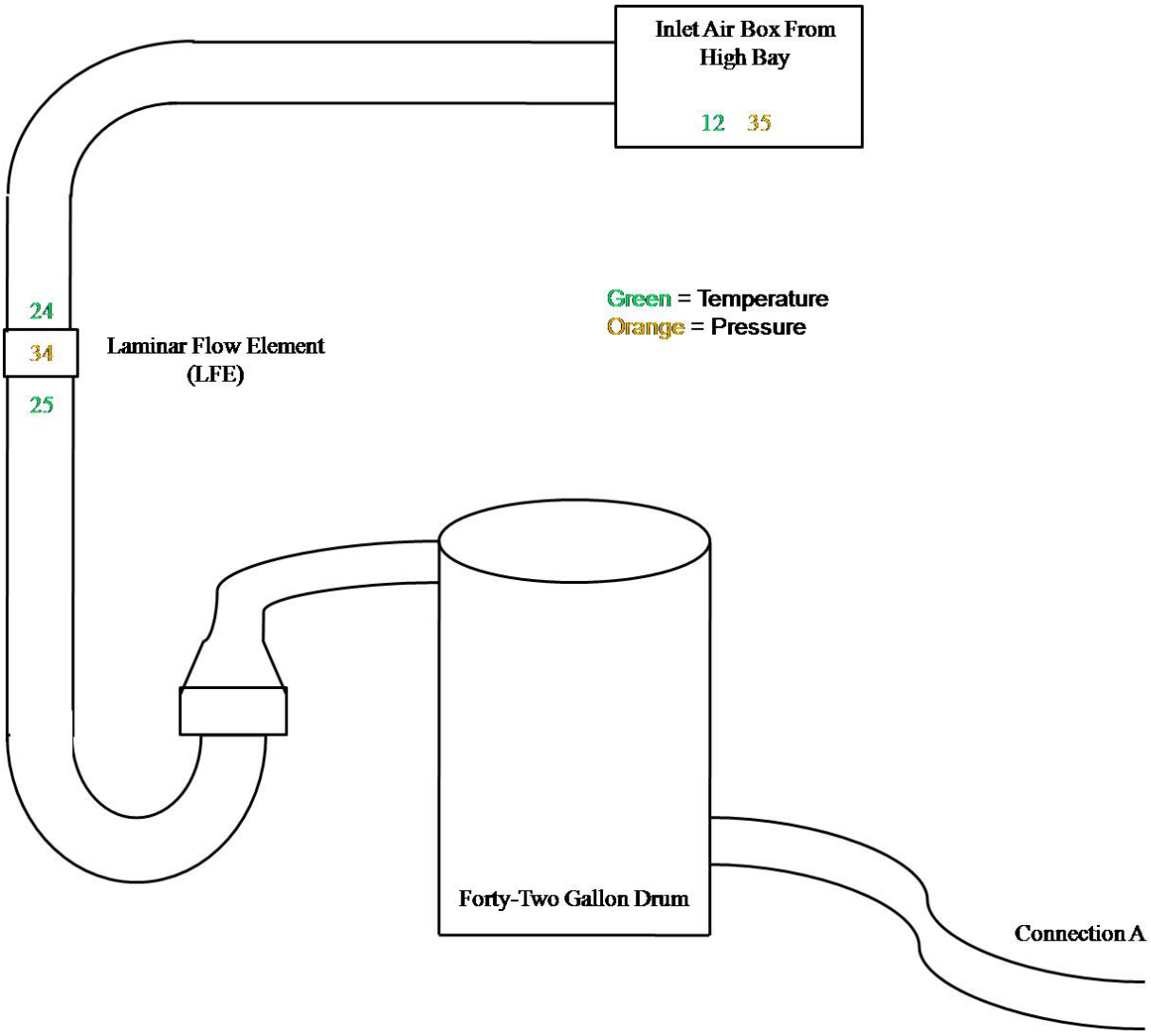


Figure 3: The system intake to bring fresh air to the engine intake system, showing the location of sensors.

Examining the sensor list, it can be seen that there were duplicate measurements for the engine speed, air fuel ratio (AFR), coolant temperature, intake manifold pressure (MAP) and intake manifold flow rate (MAF). All of these measurements were duplicates of original engine manufacturer (OEM) sensors that were recorded through the ECU. The reason for the duplication was that the OEM sensors did not have the accuracy, resolution, or measurement capabilities that were needed and these sensors were all important to the combustion model. For example the electronic control unit (ECU) air fuel ratio (AFR) was based on what the ECU thought it was injecting into the cylinder, not a measurement of what was actually injected, thus the reason for adding the universal exhaust gas oxygen (UEGO) sensor. The UEGO sensor was able to precisely measure the AFR based on oxygen present in the exhaust stream.

2.2 Data Acquisition System

To accurately capture the necessary experimental data, two data acquisition (DAQ) systems were used in the collection process, a high frequency system and a low frequency system. The high frequency system was capable of collecting data at high frequencies and used to capture the cylinder pressure data. This system operated in the CAD domain, which will be explained in further detail in the following section. The low frequency system was used to record cycled average data, or data that is essentially constant over time and was averaged over the complete recoding cycle.

The high and low frequency DAQ systems were independently configured using LabVIEW for data acquisition control. Specifically LabVIEW interfaced between the

measurement and control hardware, analyzed data and displayed the results graphically in a real time application. Even though the two systems have independent programming, they were linked together, via a serial interface, so recording of the two systems could be started simultaneously, which is a feature of the LabVIEW program.

2.2.1 Low Frequency Data Acquisition System

The low frequency DAQ system was composed of three input modules to compensate all of the low frequency measurements. Modules 1 and 2 were National Instruments NI SCXI-1100 boards and module 3 was a National Instruments NI SCXI-1102 board, which was specifically designed for thermocouple measurements. These boards were chosen due to the programmable gain and filter settings for conditioning the signals and the ability to multiplex the inputs into a single channel thus allowing the boards to be stacked to accommodate for all the channels. The low frequency DAQ system was setup to sample at 100 Hz. The data acquired with this system was cycle averaged data meaning that each acquisition was averaged over the complete sampling time, resulting in a single steady state value for each engine variable. The complete specifications for these modules can be seen in Appendix B.

The measurements that were relevant to the combustion model recorded by the low frequency DAQ system were the inlet and outlet coolant temperatures, the four intake runner temperatures, the engine speed, the differential laminar flow pressure, the intake and exhaust cam positions, the air/fuel ratio and the spark timing. The temperatures were measured using type K thermocouples. The output of the

thermocouples correlated to specific temperatures based on the Seebeck affect (Figliola and Beasley). The speed of the engine was taken from the dynamometer, in RPM, which determined the speed using an optical encoder. The intake cam, exhaust cam, air/fuel ratio, and spark timing were OEM measurements that were taken directly from the ECU.

A Sensotec FP2000 differential pressure sensor was used to measure the differential pressure across the laminar flow element. The sensor was able to read differential pressures from 0 to 10 inH_2O. Measuring the differential pressure across the LFE allowed for the mass flow rate to be calculated. A calibration was completed for the LFE to correlate differential pressures to volumetric flow rates. A least squares fit was then used for the data,

$$CFM(curve) = 35.3544 \cdot DP - .19544 \cdot DP^2 \tag{5}$$

where *CFM* was the volumetric flow rate and *DP* was the differential pressure. The volumetric flow rate was then converted to mass flow rate using the density of air. Since the density of air is a function of temperature and pressure it was calculated using the ideal gas law,

$$\frac{m}{V}(density) = \frac{p_{air}}{R_{air}T_{air}} \tag{6}$$

where p_{air} was the atmospheric pressure of the air measured during the engine test, T_{air} was the temperature of the air measured at the point where it flowed through the LFE and

R_{air} was the gas constant for air. The complete performance of the differential pressure sensor and LFE can be seen in Appendix C.

2.2.2 High Frequency Data Acquisition System

The high frequency DAQ system was composed of two input modules. The first module was a National Instruments NI PCI-6036E used for analog input from the sensors. This module was low in cost but was able to accurately sample at the high frequencies needed. The second module was a National Instruments PCI-6602, which was a counter card. The counter card was needed to protect from any false triggering in the data collection process due to any noise in the system. To accomplish this, the counter card adds a delay to the triggering signal to detect and filter any noise that might occur. This will be better understood after the encoder description later in this section. The complete specifications for these two modules can be seen in Appendix D.

The high frequency DAQ system operated in the crank angle domain instead of the time domain, meaning that samples were recorded based on the angle of the crankshaft instead of a set number or samples per second. The configuration that was used for experimentation recorded one data point for every degree the crankshaft rotated. This corresponded to 360 recorded data points per complete revolution of the engine; however the frequency at which the data was collected depended on the speed of the engine. The frequency ranged from 4.8 kHz at idle, 800 RPM, to 38.4 kHz at maximum engine speed, 6400 RPM.

The measurements collected by the high frequency DAQ system relevant to the combustion model were the four cylinder pressures, the exhaust pressure, and the crankshaft encoder. The cylinder pressure sensors were piezoelectric sensors made by AVL. This type of sensor uses a quartz crystal which sends out an electric charge that is proportional to the pressure. A charge amplifier was used to change the signal to a voltage. Piezoelectric sensors were used because they have excellent dynamic properties, were compact, and have good temperature stability. These sensors measured relative pressures, meaning that when using these sensors they must be referenced to an absolute sensor to get true pressures, and could measure over a range from 0 to 250 bar. A major advantage to these sensors was that they came mounted inside the spark plugs. The OEM spark plugs were replaced with these instrumented spark plugs which eliminated any machining to the cylinder head. Figure 4 shows the transducer mounted inside the spark plug. For further specifications on the cylinder pressure transducer refer to Appendix E.

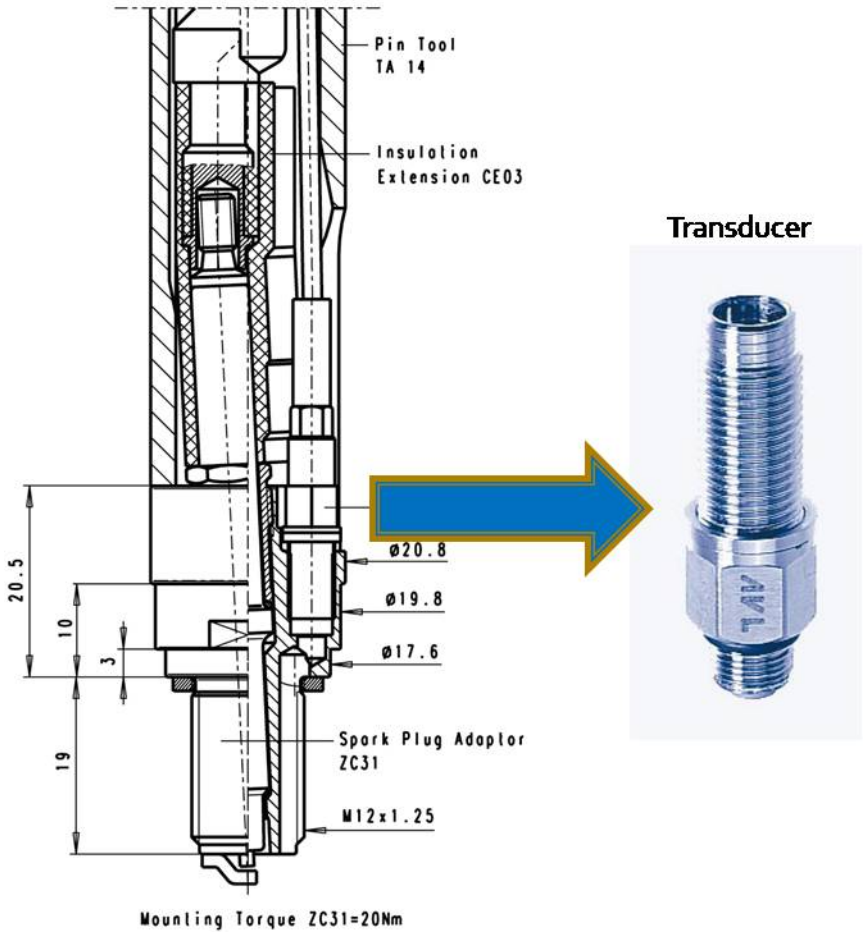


Figure 4: Piezoelectric pressure transducer mounted inside the spark plug.

The exhaust pressure sensors was a piezoresistive sensors made by Kistler. This type of sensor uses piezoresistive resistors in a Wheatstone bridge. Applied pressure unbalances the bridge and outputs proportional electric signals. The operating range for this sensor was from 0 to 2 bar, absolute pressure. A cooling jacket, which was constantly supplied with fresh water, was used to keep the sensor within its temperature

limitations since it was exposed to the high exhaust gas temperatures. For complete specifications on the exhaust sensor refer to Appendix F.

The crank angle encoder was an optical encoder made by BEI, see Appendix G for specifications. It was mechanically coupled to the engine crankshaft, using a zero backlash coupler, and was used as the triggering device for data collection. To further understand this, the functionality of the crank shaft encoder must be understood. The encoder outputs two channels to the DAQ system. The first channel sent out a pulse for every degree that the encoder rotated and the second channel output one pulse for every rotation of the encoder, when the encoder was at its top dead center (TDC) position. This TDC pulse was mechanically aligned to TDC of the first piston of the engine.

Figure 5 shows the output waveforms for these two channels. The $\overline{A}$ output signal was used as the pulse generator at every degree and the *Z* output was the TDC pulse. Any of the four channels, $A, \overline{A}$, B, or $\overline{B}$, could have been used as the output channel; it just needs to be known for shifting purposes. The $\overline{A}$ channel was used as the triggering device for data collection since a pulse was output at every degree of rotation. When the DAQ system received the encoder pulse it would record the four cylinder pressures and exhaust pressure. The DAQ system was setup to record at the rising edge of the waveform.

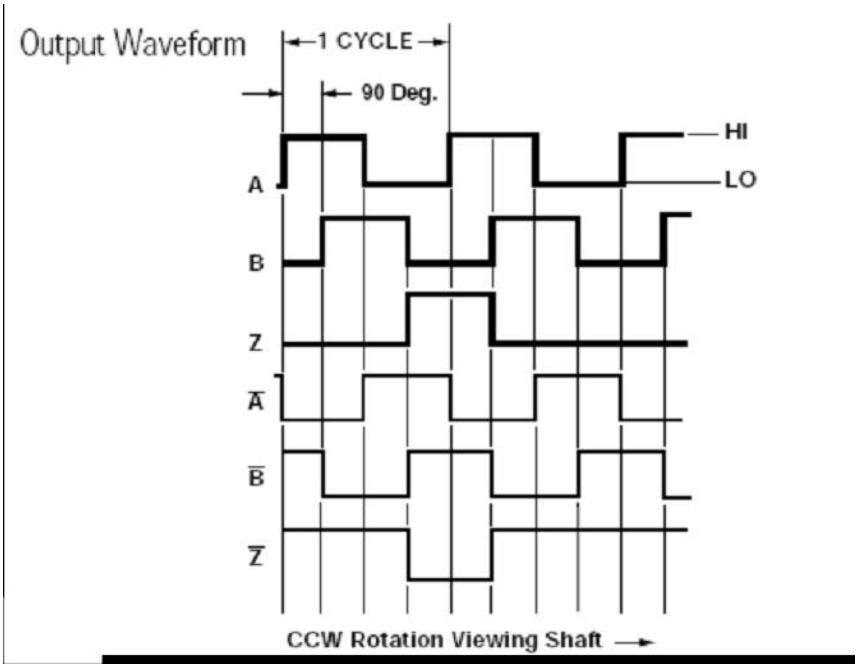


Figure 5: The pulse waveform outputs of the crankshaft encoder.

The data file would not start acquiring the signals until a TDC pulse was received by the DAQ system. However no data was collected at the TDC pulse, it just initiated the file; data was only collected at the $\overline{A}$ pulses. Therefore the data files first point occurred when the first piston was at TDC. This concept was extremely important when dealing with the cylinder and exhaust pressures during conditioning and analysis. Since the first data point in the file always occurred when the first piston was at TDC, and each data point corresponded to one degree of crankshaft rotation, then the cylinder pressures could be matched exactly with the piston location.

Once recording was initiated, the program would start acquiring the pressures at every degree of the crankshaft. However due to the characteristics of combustion and variability in the experimental setup, there were variations in cylinder and exhaust manifold pressures on a cycle by cycle basis. To limit these variations and get more precise pressure traces, 200 cycles of the engine were recorded in a continuous file so that an ensemble average could be taken. Figure 6 shows the first part of a cylinder pressure file for about 20 continuous cycles. Each of the peaks in the figure corresponds to a single combustion event.

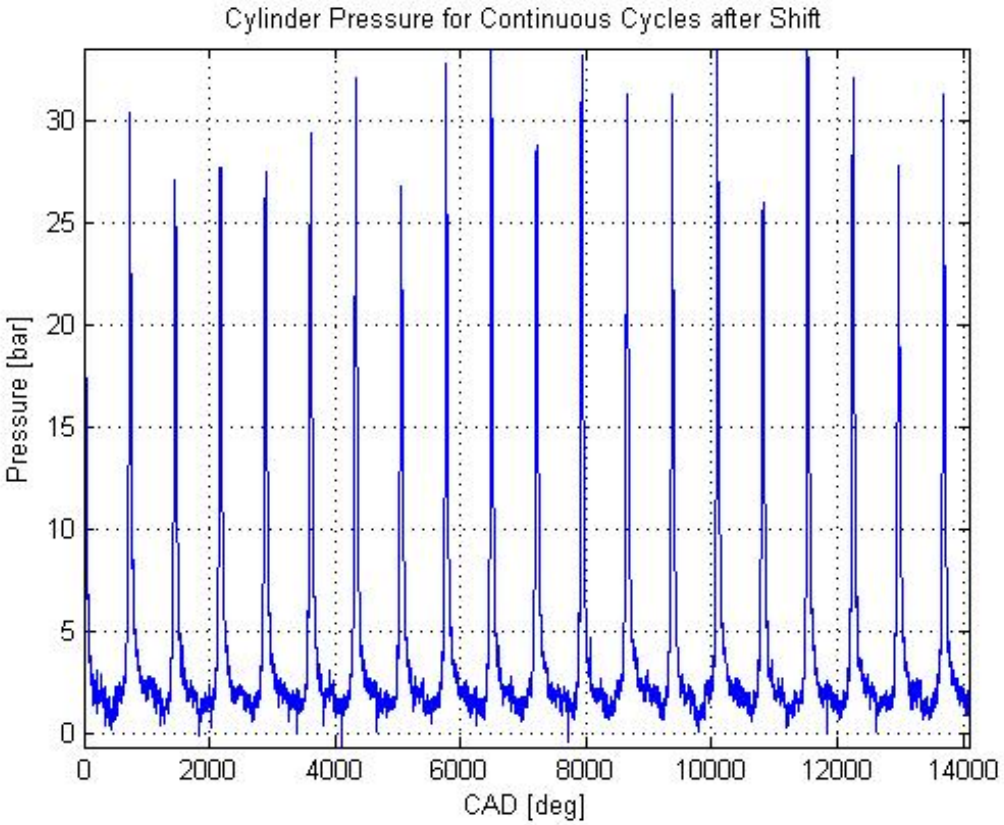


Figure 6: Example of first 20 cycles of cylinder pressure trace.

2.3 Summary

This experimental setup was used to collect data over the entire operating range of the engine, subject to changes in engine speed, throttle position, intake cam timing, exhaust cam timing, air fuel ratio, and spark timing. Three thousand operating conditions were tested using a space filling design of experiments in order to cover the operating range of the engine. The experimental data was then used in creating the following methodology and calibration for the combustion model.

CHAPTER 3

POST-PROCESSING OF CYLINDER PRESSURE DATA

Since the combustion model was an empirical model, the accuracy of the model was based on the accuracy of the data collected and the methods for transition from the raw data to burn rate profiles. In order to conquer these obstacles a systematic approach was developed. The first step of the methodology was to build a tool which processed the raw cylinder pressure data. The finalized tool was automated thus allowing large amounts of data to be processed with minimal efforts.

To use the cylinder pressure affectively and accurately in the burn rate model the pressure traces must be conditioned with the end result shown in Figure 7. This figure shows the cylinder pressure for a complete cycle of the engine, which is based on 720 degrees or two rotations of the crankshaft due to the four stroke cycle, with TDC fire at 180 degrees. In order for the cylinder pressure to become useful the raw data must go through a series of conversions, shifts, offsets, ensembles and checks. The following will describe the automated procedure that occurs when conducting the cylinder pressure conditioning and provide specific examples. This procedure was implemented into a Matlab tool which can be seen in Appendix H and Appendix I. The code in appendix H

is the user interface allowing for specific data files to be processed and the code in Appendix I actually performs the following procedures.

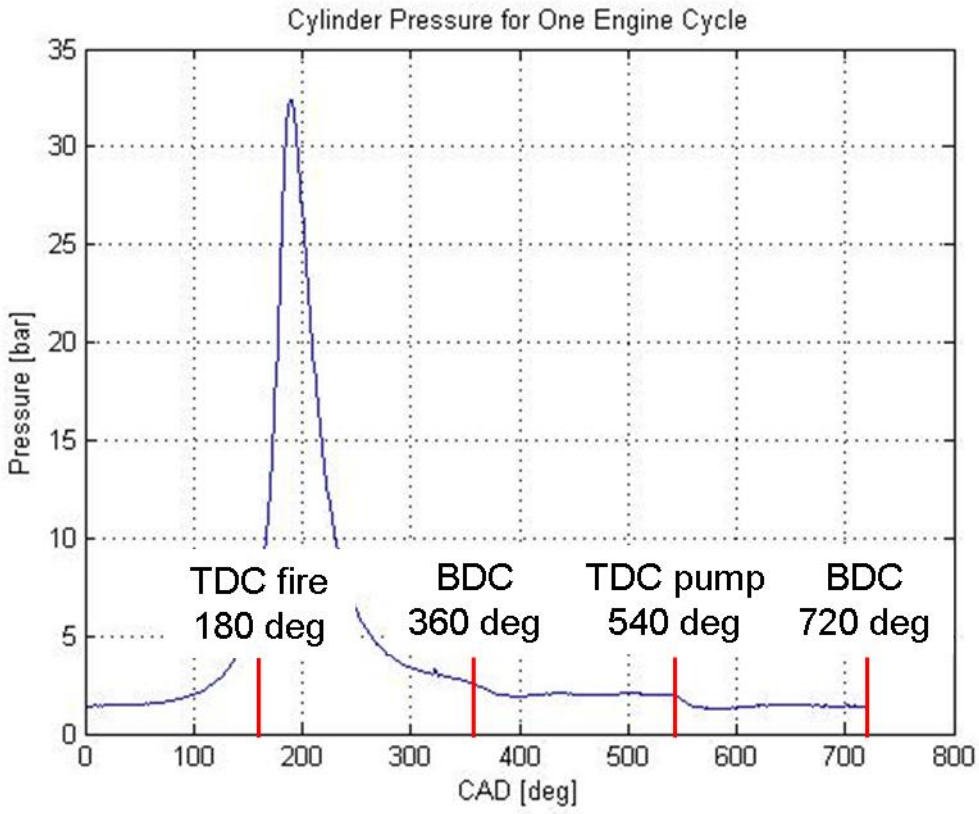


Figure 7: Conditioned cylinder pressure, with TDC fire at 180 degrees.

3.1 Low Pass Filtering and Raw Data Conversion

The first step in conditioning the raw data was to pass the cylinder pressure traces (the entire acquisition extended to 200 cycles) and exhaust manifold pressure trace through a low pass filter in order to remove any noise that was inherent in the data. In particular a forward and reverse filter was used to ensure no phase lag was associated with the filtered data. The filtered data was then converted from the raw voltage signals, from the sensors, to known pressure units. To convert to pressure units, the sensitivity, or slope, and an offset for each sensor was needed. These two values where used in equation 1 to find the pressures.

$$\mathrm{Pr}\mathit{essure}[\mathrm{bar}] = \mathit{Voltage}[\mathrm{mV}] * \mathit{Sensitivity}[\mathrm{bar/mV}] + \mathit{Offset}[\mathrm{bar}] \tag{7}$$

The sensitivity for each sensor was found from the calibration sheets from the sensor manufactures and the offset values were found using a master sensor. The offset was only needed for the exhaust pressure sensor however since it measured absolute pressure. The relative cylinder pressure sensors will have offsets applied in a different manner, which is explained in further detail in the following section. To find the offset for the exhaust pressure sensor it was compared to a master sensor at atmospheric conditions and the difference between the two sensors determines the offset.

3.2 Cylinder Pressure Pegging

To get absolute cylinder pressure values, the relative cylinder pressures were pegged to an absolute pressure, meaning that an offset was applied to the relative cylinder

pressures based on the difference between the relative cylinder pressure and an absolute pressure. Figure 8 displays the cylinder pressure for about 50 engine cycles to show how the relative cylinder pressure can drift, therefore giving reason for pressure pegging. The exhaust pressure was used as the reference value since it was an absolute pressure. To successfully accomplish this, each cycle of the cylinder pressure and corresponding exhaust pressure cycle were isolated and an offset was determined. There were 200 cycles in each cylinder pressure file, therefore for each cylinder pressure trace 200 offsets where determined. Each offset was then applied to the entire corresponding cylinder pressure cycle. Figure 9 shows the same pressure trace as Figure 8 after it has been pegged. Using pressure pegging the drift was eliminated and absolute cylinder pressure was obtained.

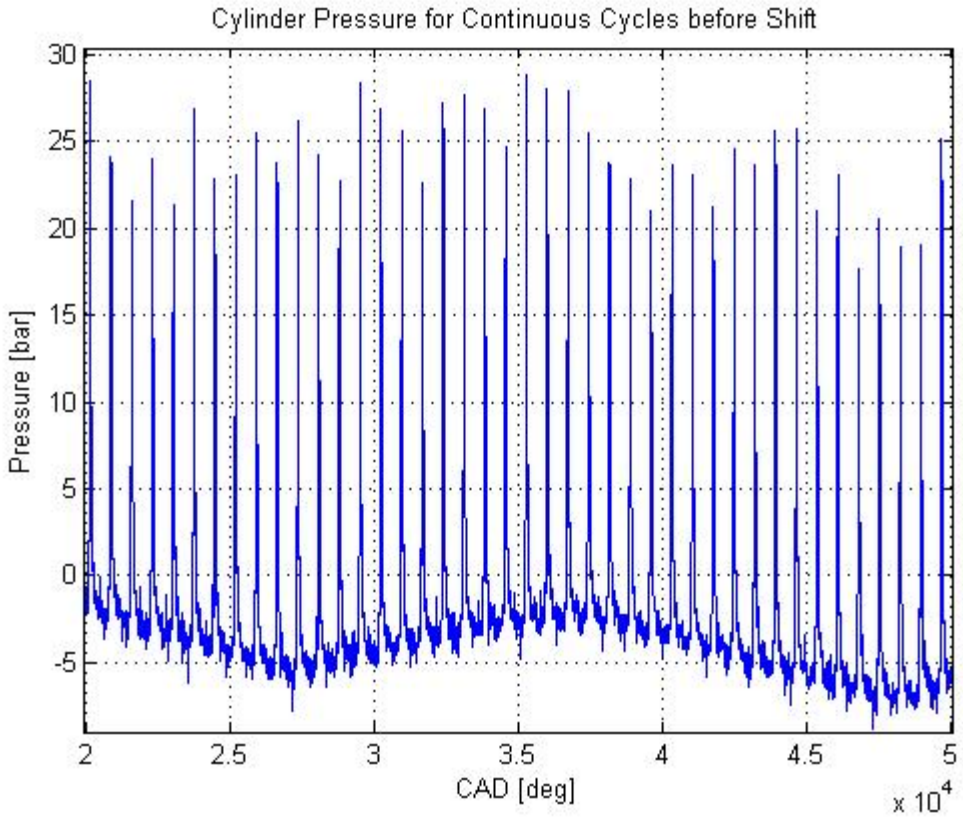


Figure 8: Cylinder pressure for about 50 engine cycles to show how the relative cylinder pressure can drift.

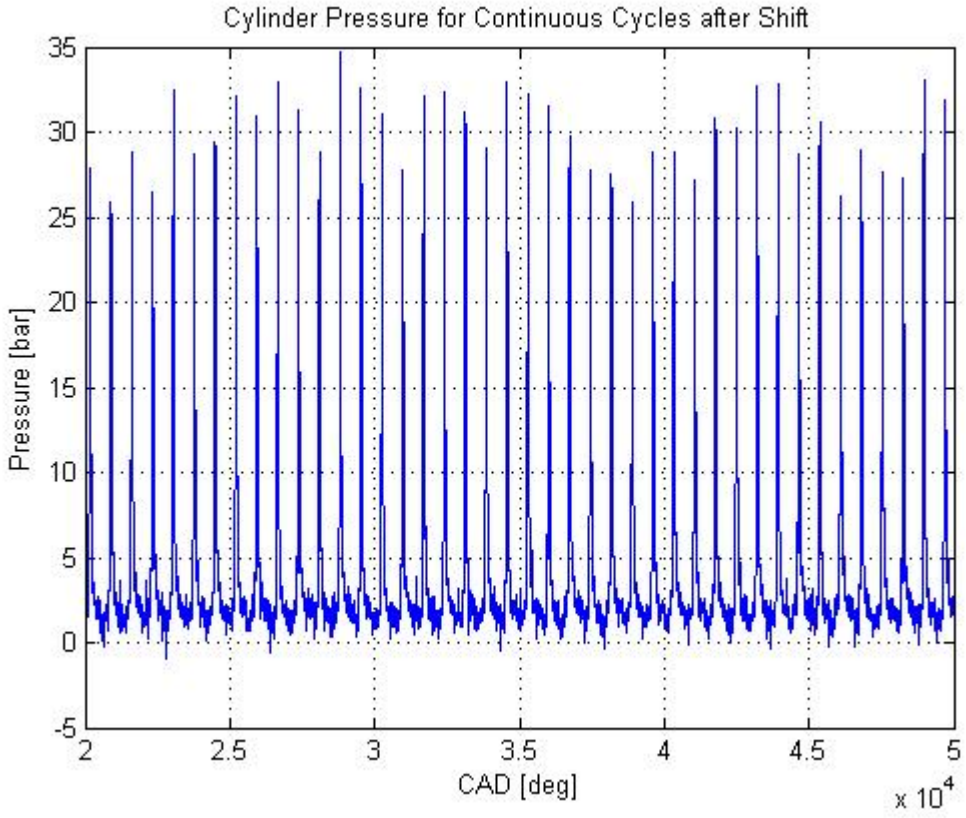


Figure 9: The cylinder pressure trace from Figure 8 after it has been pinned to the exhaust pressure.

The offsets where determined for each cycle by taking an average over an interval during the exhaust stroke. Figure 10 shows a single cycle for the cylinder pressure and the corresponding exhaust pressure during the cycle. The pressures traces were oriented to have the piston top dead center (TCD) during the firing stroke at 180 degrees. Therefore, the exhaust stroke occurs between 360 and 540 degrees. The interval where the offset was calculated is shown on the two figures and occurred between 440 and 480 degrees. This interval was used for all offset calculations. The average is taken over the

interval for both the pressure traces and the offset was determined as the difference between the two. From these two figures it can be seen that the offset is about 1.5 bar which is a significant difference; therefore this is a very important step in the pressure conditioning process.

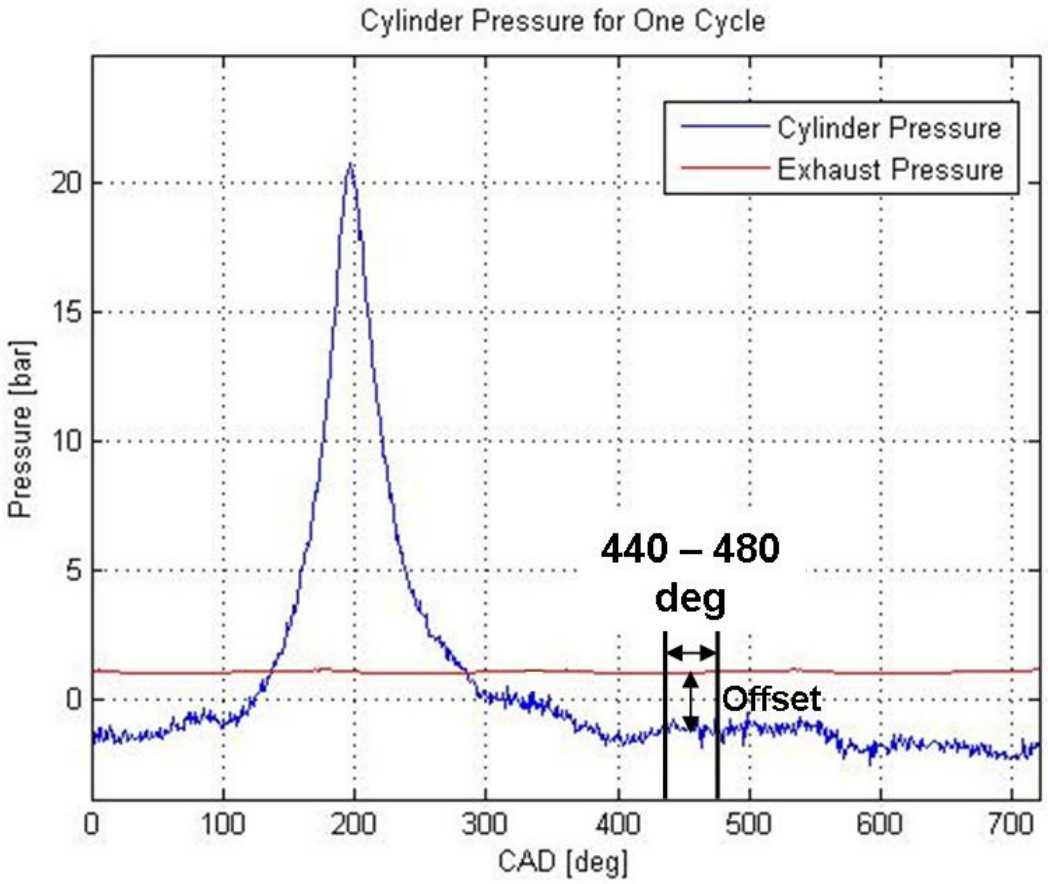


Figure 10: Cylinder pressure and exhaust pressure for one cycle to show pinning process and determination of the offset.

This method for pegging the cylinder pressure was chosen based on the research found from the literature review. The engine being modeled for this project had a tuned intake system thus excluding the first pressure pegging method, which pegged the cylinder pressure to the intake pressure. This left the polytropic method and the exhaust pressure method. The exhaust pressure method was chosen because it was proven to be a reliable method in cylinder pressure pegging due to the miniscule pressure fluctuations. The range that was chosen was in the middle of the exhaust stroke after the blow down process and before intake valve opening, to ensure a range where the cylinder pressure is equal to exhaust backpressure.

3.3 Firing Order Shift

To get each of the cylinder pressure traces to have TDC fire at 180 degrees each had to be shifted according to the firing order of the engine. It was extremely important than any shifting that occurred to the cylinder pressure trace also occurred to the exhaust pressure trace to ensure that they corresponded to each other. To keep track of the shifting for corresponding cylinder pressure traces and exhaust pressure traces an exhaust pressure trace was assigned to each cylinder pressure, even though the exhaust pressure would be the same for each cylinder. This allowed for shifting to be conducted on both cylinder and exhaust pressures simultaneously.

The engine used for this project was a four cylinder, four stroke engine meaning that a piston fired every 180 degrees. The firing order for the engine was 1-2-4-3. It was initially assumed that the DAQ system was initiated by the TDC pulse during the firing stroke of the first piston. Figure 11 shows the orientation for all four cylinders based on this assumption, before any shifts occur, for a complete engine cycle. In order for all the cylinder pressures to be oriented with TDC fire located at 180 degrees, the cylinder pressure files had to be shifted according to Table 3. A circular shift was used, meaning that the data was shifted circularly, with the data from the beginning of the file going to the end, or vice versa. So the first cycle and last cycle, cycle 200, ended up conjoining to form a cycle. This cycle would have to be discarded due to inaccuracies associated with conjoining cycles before the assemble average.

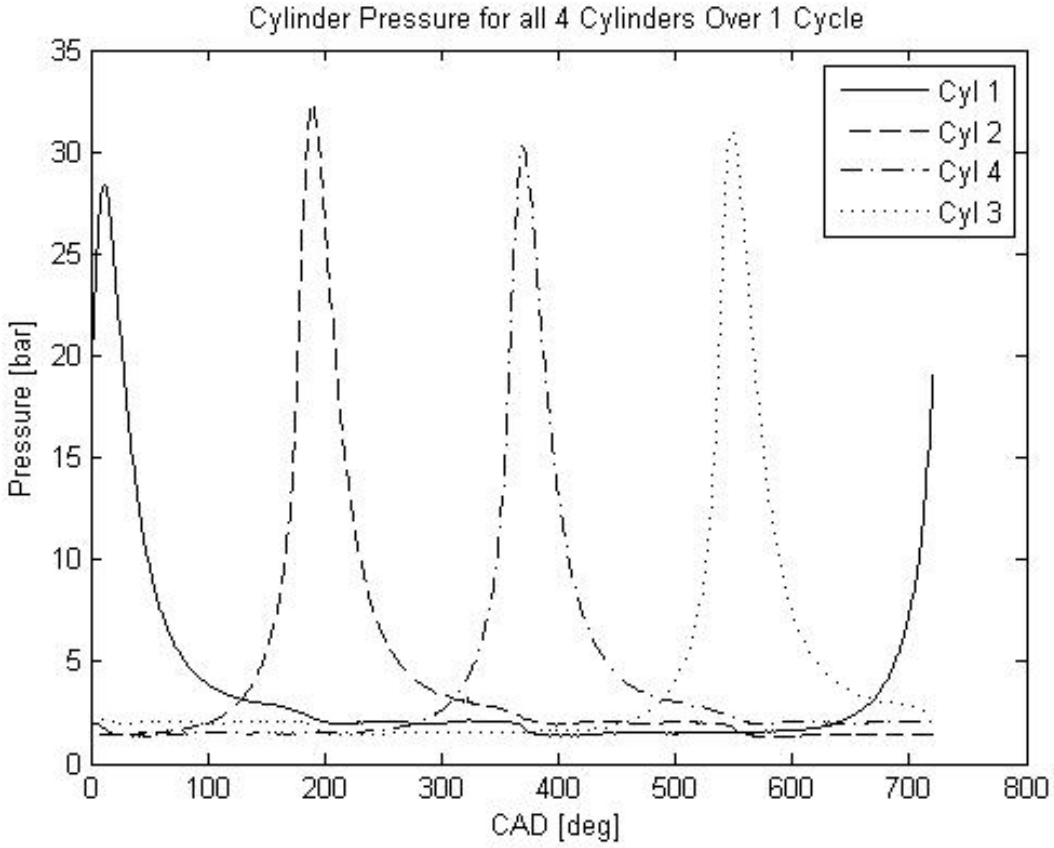


Figure 11: Cylinder pressure over one complete engine cycle for all four cylinders before any shifting occurs.

Table 3: Describes the shifts needed for each cylinder to achieve TDC fire at 180 degrees.

Cylinder Number	Shift [deg]
One	180
Two	0
Three	-180

3.4 Ensemble Average

The final step in conditioning the cylinder pressure data was to average the 200 engine cycles to get one ensemble engine cycle. This was completed for all four of the cylinders. The ensemble average for each of the four cylinders was then averaged, thus the reason for shifting them to all have TDC fire at 180 degrees. Figure 12 shows the ensemble average for one cylinder. In comparison to Figure 10, which is a single cycle, the pressure trace is much smoother. The ensemble average is affective in eliminating any noise inherent in the signal, which was not filtered, and creates a good approximation for a cylinder pressure during one cycle. The word approximation was used because the cylinder pressure will be different for every cycle, due to combustion characteristics, however it gives a good approximation. Some characteristics that affect the combustion include the air fuel ratio, fuel composition, mixing, temperatures, and residuals.

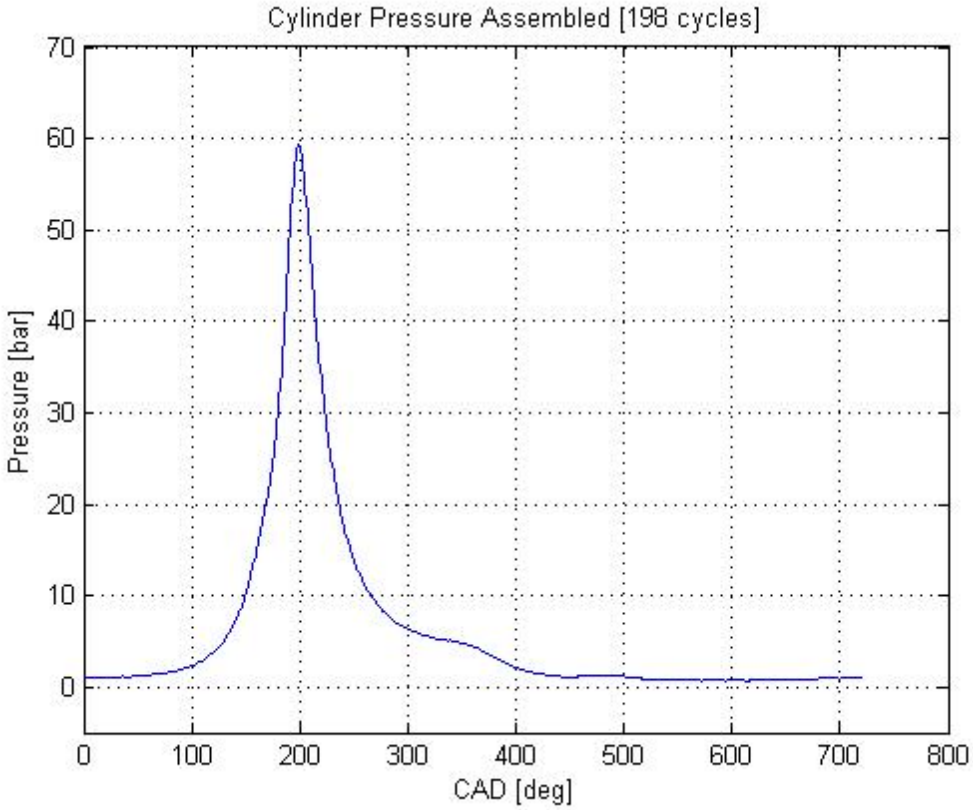


Figure 12: Cylinder pressure after the assembled average of 198 cycles.

3.5 Common Issues in Cylinder Pressure Data Acquisition

When conditioning the cylinder pressure data some common issues were encountered and solutions to these problems had to be integrated into the post-processing methodology. Solving these issues was essential when processing the cylinder pressure data since the cylinder pressure was the used directly for the combustion model. The following will describe each of the issues encountered and the solutions used in solving the problems.

3.5.1 Missing TDC Pulse

During the cylinder pressure post processing an initial assumption was made that the DAQ system started recording based on the TDC pulse from the firing stroke of the first piston. However it was possible that the DAQ system could be initiated by a TDC pulse from the pumping stroke of the first piston. If the initial assumption was wrong and in fact the DAQ system was initiated by the TDC pulse from the pumping stroke, then after the cylinder shifts due to firing order, the TDC fire would be located at 540 degrees for each of the four pistons. The cylinder pressures would be 360 degrees out of phase caused by the fact that TDC of the firing stroke and pumping stroke are 360 degrees out of phase. Figure 13 shows the resulting cylinder pressure due to this z-index error. This problem was corrected by isolating the first few cycles of the first cylinder and looking where the maximum pressure occurs. If the maximum pressure was around 180 degrees then no shift occurred, but if the maximum pressure occurred around 540 degrees then the entire pressure trace was shifted 360 degrees. This again used a circular shifting scheme, therefore causing the conjoining of the first and last cycle. These cycles would need to be removed before the assemble average. The z-index error causes all four cylinders to be out of phase, not just one, so all four cylinders were shifted together if needed.

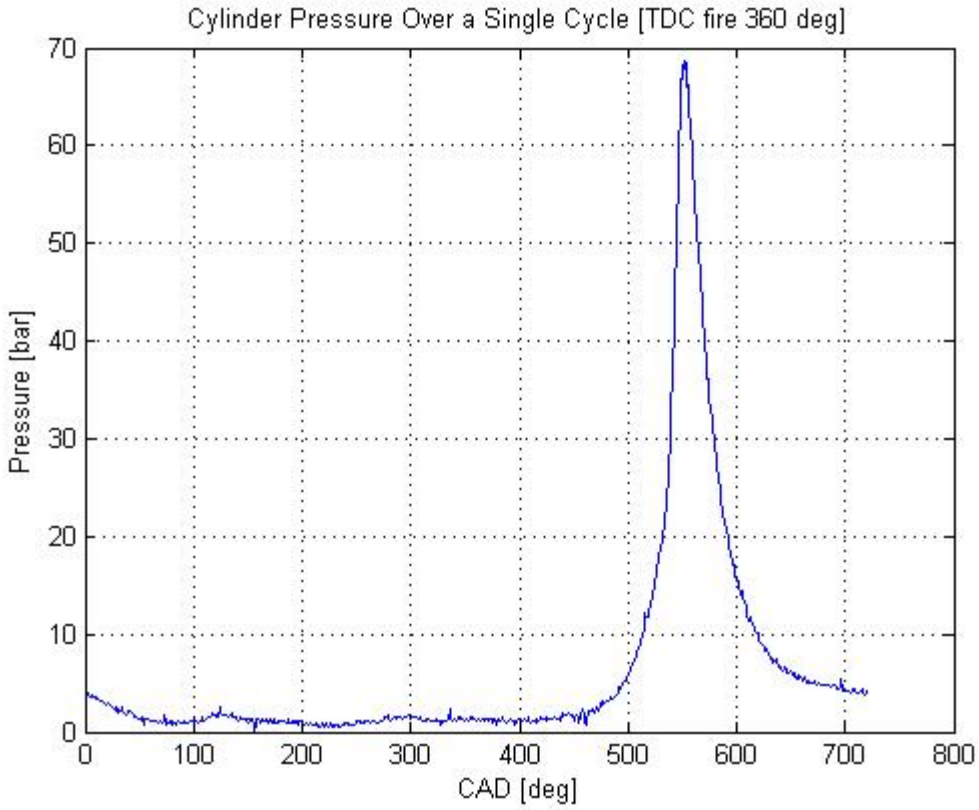


Figure 13: Cylinder pressure showing the result of recording initiated by a TDC pulse during the pumping stroke.

3.5.2 Encoder Alignment Error

The TDC pulse of the crankshaft encoder was mechanically aligned to TDC of the first piston during the engine instrumentation and setup phase of the project. The tools and instrumentation used during the alignment allowed for an accuracy within .1 CAD. After alignment the CAD error between the encoder TDC and piston TDC could be measured using the setup equipment. It was determined that the encoder's TDC pulse

was .1 degrees after the true TDC of the piston. This causes all the pressures traces, cylinder and exhaust, to be delayed .1 degrees from actual mechanical alignment. The solution for this error will be addressed in Section 3.5.5.

3.5.3 Encoder Time Delay

The second phasing error was a time delay associated with the DAQ counter card. The DAQ counter card would receive the pulse from the encoder but would delay a fixed time, $10.7e^{-6}$ seconds before recording a data point. The time delay was implemented to ensure that false triggering did not occur due to any noise associated with the encoder output signal. If extra data was added to a pressure trace, then cycles would not consist of 720 data points, or one for every degree, which would cause errors during processing. However, this time delay also delays the pressure traces from actual mechanical alignment with the pistons. To compensate for this error it first had to be transformed from the time domain to the crank angle domain. Since the time delay was a fixed time, the crank angle degrees associated with the error would vary depending on the speed of the engine. Equation 2 was used to transfer the time delay into crank angle error based on the engine speed. The actual shift will be addressed in Section 3.5.5.

$$Delay[\text{deg}] = 6 \cdot Delay[\text{s}] \cdot N[\text{RPM}] \tag{8}$$

3.5.4 Encoder Phase Lag

The third and final phasing error associated with the encoder dealt with the delay between the TDC pulse of the encoder and the pulse that actually records a data point.

The TDC pulse, z output, was used for the mechanical alignment, but the actual TDC data point was not recorded until the first $\overline{A}$ pulse after the TDC pulse was received. Figure 5 shows the output cycles of the encoder, in which it can be seen that the $\overline{A}$ output trails the Z output by .75 of a cycle. Since one cycle corresponds to 1 degree then there is a .75 degree delay between the time when the piston is a TDC to when an actual data point was acquired.

3.5.5 Total Encoder Error

All three of the phasing errors associated with the encoder, the encoder alignment error, the encoder time delay, and the encoder phase lag, delayed the pressure traces form mechanical TDC. This means that the piston was actually past TDC by the sum of these three errors, in CAD, for the data points corresponding to TDC. This also meant that every subsequent point also had this same delay error. In order for the data to correspond exactly to the mechanical location of the piston all the pressure files must be delayed by the total encoder error. When shifting the data due to this encoder error, a circular shift could not be used because the shift was not a multiple of a degree. The circular shift could only be used when shifts were at an exact degree, no fractions of a degree, because every data point corresponded to a single degree so a shift to the next data point corresponded to a one degree shift. Instead interpolation between the data points was conducted based on the calculated encoder error. Figure 14 shows an exploded view of the cylinder pressure at peak pressure during combustion. This is the first cycle of the cylinder pressure trace and shows the pressure before and after the encoder error shift.

The shift is very small but necessary for complete accuracy. The shift now allows for the data files to correspond exactly with the mechanical position of the piston, with TDC fire exactly at 180 degrees.

The total encoder error could range from .85 degrees at essentially zero engine speed to 1.26 degrees at maximum engine speed, 6400 RPM, due to the variable encoder error associated with the DAQ time delay. When using interpolation to shift the data the first data point was lost in the interpolation if the error was less than one degree and first two data points were lost if the error was greater than one. This can be seen in Figure 14, the shifted cylinder pressure lost the first two data points.

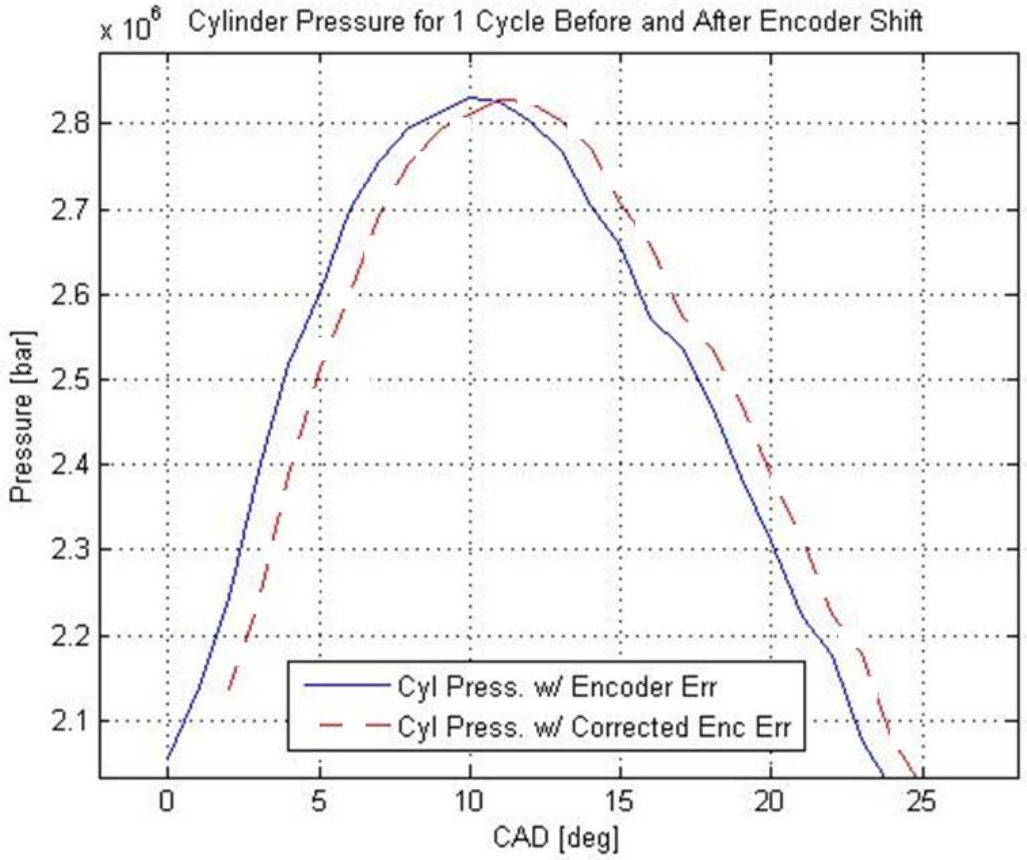


Figure 14: Exploded cylinder pressure trace showing the shift that occurs due to the encoder error.

At this stage the cylinder pressures and exhaust pressures were all shifted to have TDC fire exactly at 180 degrees. It was mentioned during each of the shifts the affects that the shifts had on the first and last cycle in the pressure traces. Since all shifting is completed, the first and last cycle in the pressure traces were now removed from the files since they contained inaccurate and incomplete data. The removal of these cycles left the pressure files with 198 cycles instead of 200 cycles.

3.6 Summary

The first stage of the methodology was successful in transforming the raw cylinder pressure data into a single conditioned cycle of the engine. The four main steps in conditioning the cylinder pressure data consisted of filtering and converting, cylinder pressure pegging, shifting with respect to firing order and the ensemble average. Further steps were taken to ensure that TDC fire was located exactly at 180 degrees. These steps were related to the experimental setup and instrumentation used, however were necessary steps to get the accuracy desired for conditioned cylinder pressure.

CHAPTER 4

DATA DIAGNOSTICS

To ensure that the experimental data was accurate, diagnostics were developed to check the data throughout post-processing. This was implemented as an automated check due to the large data set that was collected for the project to reduce user interaction time. These diagnostics were incorporated into the tool developed for the post processing which is shown in Appendix H and Appendix I. If there were any possible errors with the data, the program would notify the user and further investigation was taken to determine the source of the problem. This section will cover the diagnostics that were developed and implemented into the automation process.

4.1 Data Acquisition Lag or Slip

Before the cylinder pressure was ensemble averaged each individual cycle was isolated and the location of the maximum pressure was determined in CAD. This data was used to detect if there was any slip occurring between the encoder and crankshaft and to determine if the DAQ system was lagging behind in data collection from an overload of data due high engine speeds. Figure 15 shows the location of the maximum pressure for each individual cycle and a linear curve fit through the data. The linear curve fit was

used to determine if the encoder was slipping because essentially every cycle should have the exact same point of maximum pressure and therefore give the linear fit a zero slope. As can be seen there was variations between cycles, which was expected due to combustion characteristics, however the slope of the line should still be relatively zero. A small tolerance was acceptable to the conditions where combustion was unstable and therefore caused larger variations, conditions with large residuals. However if there was slip between the encoder and crankshaft the maximum pressure would continually decrease at a noticeable slope. Therefore any noticeable slope would cause an error to be detected in the data.

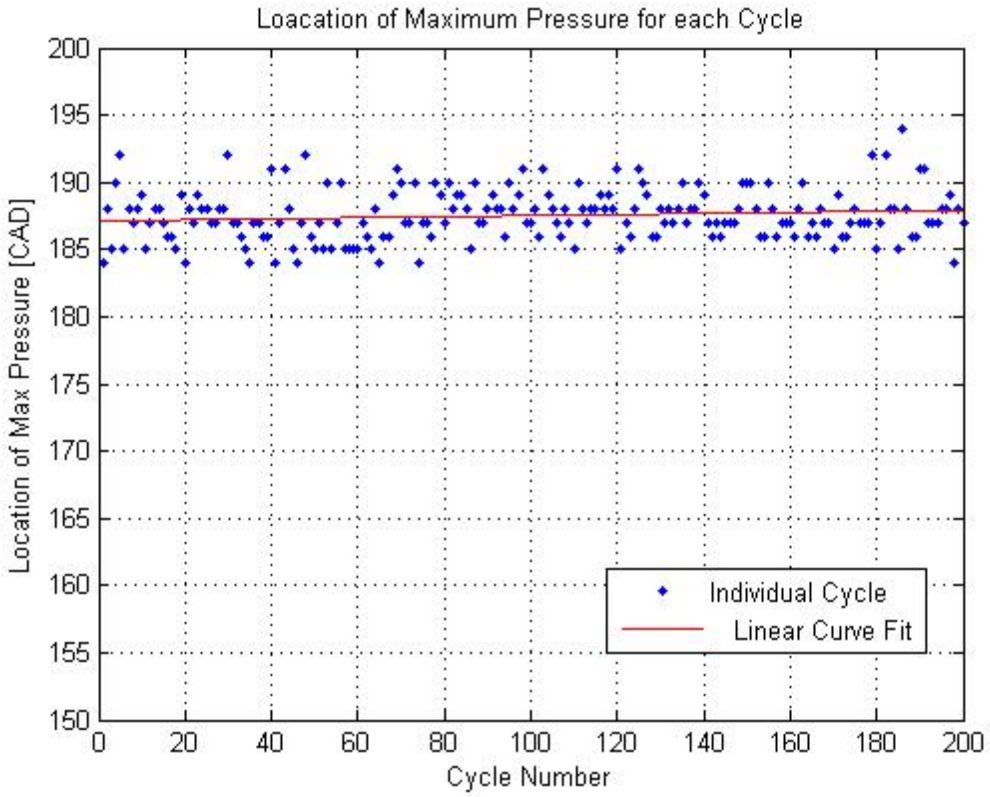


Figure 15: The location of the maximum pressure for each cycle before the assembled average with a linear curve fit through the results.

At high engine speeds, over 5000 RPM, the DAQ system became overloaded with incoming data and would lose track of data points. To detect this problem the location of the maximum pressure was used again. Figure 16 shows the results of this DAQ lag. There is a jump in the maximum pressure location due to data points being lost. The system is able to recover but then overloads again. The DAQ lag occurs towards the end of the data files and to compensate for this error the average of the maximum pressure locations for the first 50 cycles are compared to each individual cycle. If a significant

jump was found for a particular cycle, which corresponded to DAQ lag, then that cycle and all proceeding cycles were excluded from the data set and only the remaining data was used for post processing. However it was decided that at least 50 cycles were needed to get an accurate ensemble average, therefore if this problem was encountered before 50 engine cycles then an error was reported to the user.

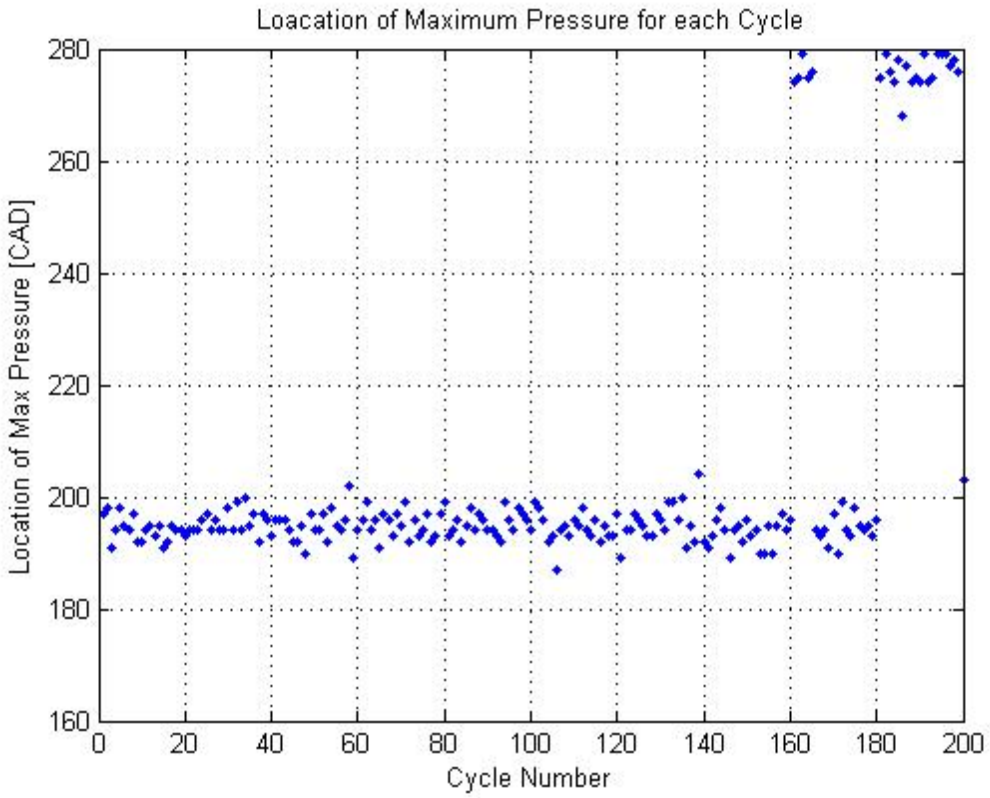


Figure 16: The location of the maximum pressure for each individual cycle showing the result of lag in the DAQ system.

4.2 Cylinder and Exhaust Pressure Checks

Once the ensemble average was completed for all the cylinder pressures and the exhaust pressure, the maximum and minimum pressures were determined for each pressure. A logically determined set of limits were placed on the maximum and minimum values for the cylinder pressures. For the cylinder pressures maximum value the upper limit was set based on the maximum pressure possible in the engine and the lower limit was set at the pressure which is reached during a pumping cycle. For the minimum cylinder pressure, the lower limit was set at zero pressure because it is impossible to have a negative pressure, but a vacuum does occur during the intake stroke, and the upper limit was set at atmospheric pressure because during the intake stroke the pressure has to be lower than atmospheric. Also the pressure difference was taken between the maximum pressures for the four cylinders. Differences were expected due to combustion characteristics but if large errors were present it was detected. For the exhaust pressure a range was determined based on pressures that are possibly achieved in the exhaust stroke. The upper limit was determined through experimentation on the engine at high speed under full load and the minimum pressure was set at atmospheric pressure since the exhaust should always be greater due to the flow of exhaust out the tailpipe. All these limits can be seen in Table 4.

Table 4: Cylinder pressure and exhaust pressure limits used for diagnostics.

	Lower Limit [bar]	Upper Limit [bar]
Cylinder Pressure		
Max Pressure	4	100
Min Pressure	0	1
Max Cyl. Difference	no limit	10
Exhaust Pressure		
Range	0.95	1.8

4.3 MEP Checks

The indicated mean effective pressure, IMEP, and the brake mean effective pressure, BMEP, where calculated and compared. The IMEP is the effective pressure inside the cylinder and can be calculated using the following equation,

$$IMEP = \frac{pdV}{V_d} \tag{9}$$

where p is the cylinder pressure, V is the cylinder chamber volume for a single cylinder and V_d is the displacement volume. The BEMP is the effective pressure that occurs outside the cylinder and was calculated using the following equation,

$$BMEP = \frac{\pi T}{V_d} \tag{10}$$

Where T is the torque measured by the dynamometer and V_d is the displacement volume for a single cylinder. These two values should be relatively close because the only

difference is that the BMEP includes frictional and pumping losses. Therefore it was used as a quick check to determine if the cylinder pressures where accurate because torque measured from the dynamometer is a reliable measurement.

Limits were also set for the covariance and maximum and minimum values for the IMEP. The covariance of IMEP is just the standard deviation of IMEP, using each individual cycle compared to the ensemble averaged cycle. A covariance greater than 10 percent (Heywood) meant the combustion was unstable and the cylinder pressure would not be useful for model calibration. This was seen at operating conditions with large valve overlap which corresponds to high amounts of residuals in the mixture. The lower IMEP limit was set at zero and the upper limit was set to 17, which was determined through experimentation and average IMEP values for typical SI engines (Heywood).

4.4 Summary

The data diagnostics were implemented to check the experimental data during the post processing stages from chapter 3. The purpose of the diagnostics was to notify the user if there were potential errors within the data. This was a tremendously useful feature because the program does the data processing and checks for errors instead of the user spending time looking at the large data set. Only when it was necessary was there intervention and further investigation by the user. This drastically reduced data processing time and kept bad data from being used in the model calibration.

CHAPTER 5

BURN RATE ANALYSIS

The second stage in determining the combustion model takes the conditioned cylinder pressure data and calculates the experimental burn rate, also known as mass fraction burned. A Matlab tool was developed for this stage of the combustion model which can be seen in Appendix J and Appendix K. This tool incorporated some previous work completed by Dr. Marcello Canova and would be used after the data had undergone the post-processing tool. After the user specified the data to be processed the rest of the procedure was automated. This section will cover the methods incorporated into this tool by providing specific examples and derivations.

The burn rate is the cumulative heat release that occurs during combustion (Ferguson and Kirkpatrick). When normalized the burn rate becomes a measure of combustion, ranging from zero to one, with zero corresponding to no combustion and one corresponding to complete combustion, thus meaning no more heat will be released. Figure 17 shows the three stages to get from the cylinder pressure to the normalized burn rate. The middle stage is the heat release rate which was determined using the inverse thermodynamic model(Heywood), which will be explained in further detail in the

following section. The figure shows how burn rate starts at zero and ends at one when combustion is completed.

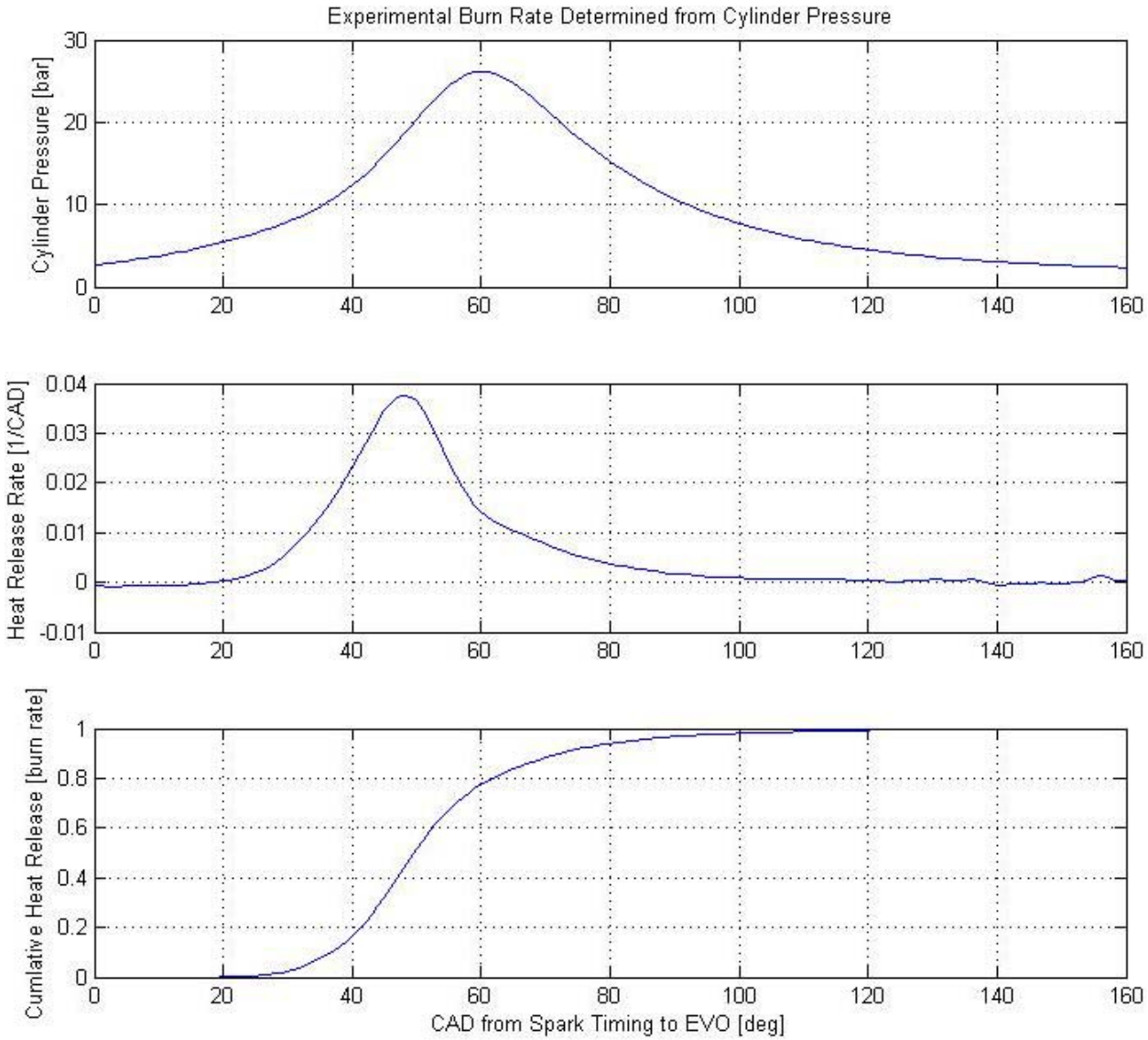


Figure 17: Using cylinder pressure to find the heat release and burn rate.

When using the inverse thermodynamic model a single zone combustion model was assumed. With this type of model only one control volume is considered in which both unburned and burned gasses are mixed at a homogeneous temperature and pressure. The following sections will derive the inverse thermodynamic model using single zone combustion. The first derivation assumes constant specific heat while the second derivation assumes variable specific heat.

5.1 Constant Specific Heat Inverse Thermodynamic Model

The control volume was considered a closed system because during combustion both the intake and exhaust valves were closed with the assumption that there was no leakage from the combustion chambers. Figure 18 shows the control volume used for the derivation. With this assumption the initial thermodynamic relationship takes the form of equation 5,

$$\frac{d\left(m_{cyl}u\right)}{dt} = \dot{Q}_{cv} - \dot{W}_{cv} \tag{11}$$

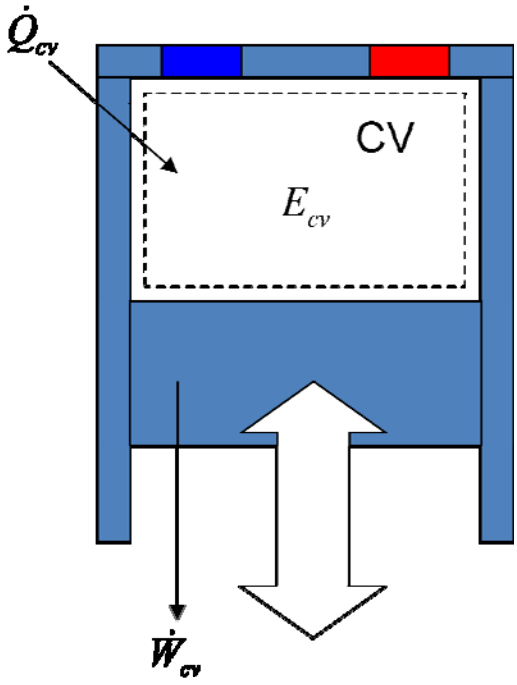


Figure 18: Control volume used for reverse thermodynamic model, with both the intake and exhaust valves closed.

where m_{cyl} was the total mass within the cylinder, including fuel, air and exhaust, u was the internal energy of the system, Q_{cv} was the heat term and W_{cv} was the work term. The work term was based solely on the work due to pressure,

$$\dot{W} = p_{cyl} \frac{dV_{cyl}}{dt} \tag{12}$$

where p_{cyl} was the conditioned cylinder pressure and V_{cyl} was the corresponding cylinder volume. The cylinder volume was determined using the engine dimensions with the relationship,

$$V_{cyl} = V_c + \frac{V_c}{2}(r_c - 1)\left(R + 1 - \cos\theta - \left[R^2 - (\sin\theta)^2\right]^{1/2}\right) \tag{13}$$

where V_c was the clearance volume, r_c was the compression ratio and R was the ratio of the connecting rod length to the crank radius. The heat term can be split into the heat released due to the chemical release from the fuel and the heat transfer that occurs through the cylinder walls, giving equation 5 the following form.

$$\frac{d(m_{cyl}u)}{dt} = -\dot{Q}_{HT} + \dot{Q}_{chemical} - p_{cyl}\frac{dV_{cyl}}{dt} \tag{14}$$

The internal energy can be related to specific heat through the relationship:

$$du = c_v dT \tag{15}$$

Since the system was considered to be a closed system, the mass can be considered constant so,

$$\frac{d(m_{cyl}u)}{dt} = m_{cyl}c_v\frac{dT_{cyl}}{dt} \tag{16}$$

Using the ideal gas law,

$$p_{cyl}V_{cyl} = m_{cyl}RT_{cyl} \tag{17}$$

and the assumption that the mass, gas constant and specific heat do not change with time then,

$$\frac{dT_{cyl}}{dt} = \frac{1}{m_{cyl}R}\left(p_{cyl}\frac{dV_{cyl}}{dt} + V\frac{dp_{cyl}}{dt} \right) \tag{18}$$

The ideal gas constant can also be represented as,

$$R = c_p - c_v \tag{19}$$

Combining equations 8, 10, 12 and 13,

$$\frac{c_v}{c_p - c_v}\left(p_{cyl}\frac{dV_{cyl}}{dt} + V\frac{dp_{cyl}}{dt} \right) = -\dot{Q}_{HT} + Q_{chemical} - p_{cyl}\frac{dV_{cyl}}{dt} \tag{20}$$

Rearranging and using $\gamma = \frac{c_p}{c_v}$ equation 14 becomes,

$$\dot{Q}_{chemical} = \frac{1}{\gamma - 1}V_{cyl}\frac{dp_{cyl}}{dt} + \frac{\gamma}{\gamma - 1}p_{cyl}\frac{dV_{cyl}}{dt} + \dot{Q}_{heat} \tag{21}$$

The final step is to switch to crank angle domain,

$$\frac{Q_{chemical}}{d\theta} = \frac{1}{\gamma - 1}V_{cyl}\frac{dp_{cyl}}{d\theta} + \frac{\gamma}{\gamma - 1}p_{cyl}\frac{dV_{cyl}}{d\theta} + \frac{Q_{heat}}{d\theta} \tag{22}$$

This final equation can be used to find the heat release based upon knowledge of the cylinder volume and experimental cylinder pressure data. This equation can be seen in combustion books like (Heywood) and (Heywood) and is known as the inverse thermodynamic model. The cumulative heat release or burn rate is just the integral of the heat release.

5.2 Variable Specific Heat Thermodynamic Model

The previous derivation was shown to give a background to the method used for this project. A single zone model was used for this project however the specific heat was not kept constant but was a function of temperature and composition. Figure 19 (Heywood) shows how the temperature, equivalence ratio and burned mass fraction affect the specific heat ratio with respect to a mixture of gasoline, air and exhaust. A significant difference can arise in the specific heat ratio, possibilities of up to 10 percent, and thus the reason to use a model that allows for variable specific heats. Figure 20 shows the difference between using a constant specific heat model and a variable specific heat model. The results show that the constant specific heat model is not accurate enough to predict the burn rate. The figure also shows the specific heat as it changes in the variable specific heat model. The shape of the specific heat was due to the combination of the temperature rising from compression and combustion and from the mixture composition changing throughout the combustion process. The specific heat begins to slope off before combustion is completed due to the temperature decreasing within the cylinder during the expansion stroke.

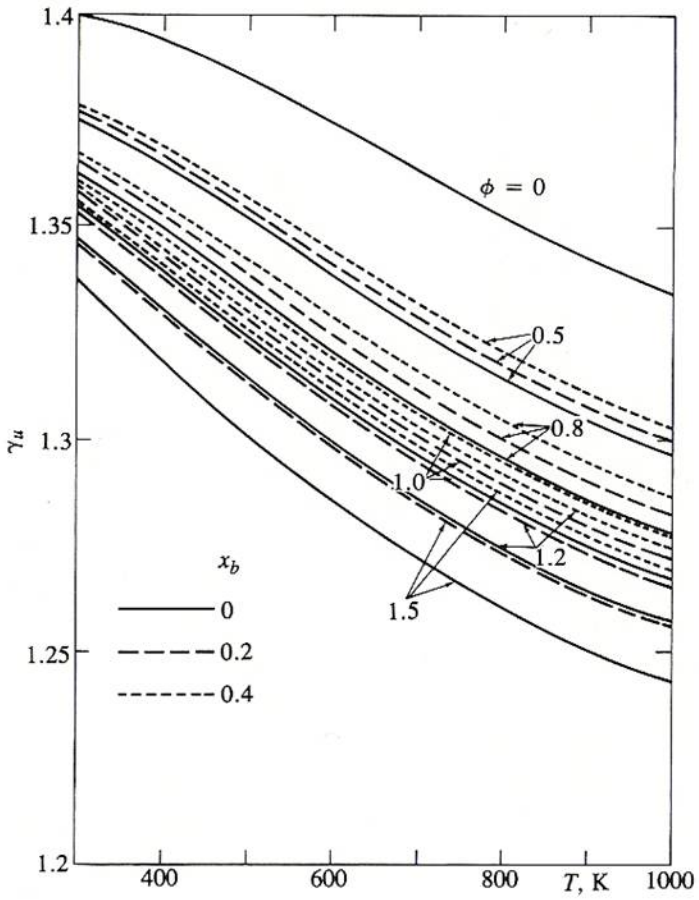


Figure 19: Ratio of the specific heats, $\gamma = c_p / c_v$, or unburned gasoline, air, burned gas mixtures as a function of temperature, equivalence ratio, and burned gas fraction.

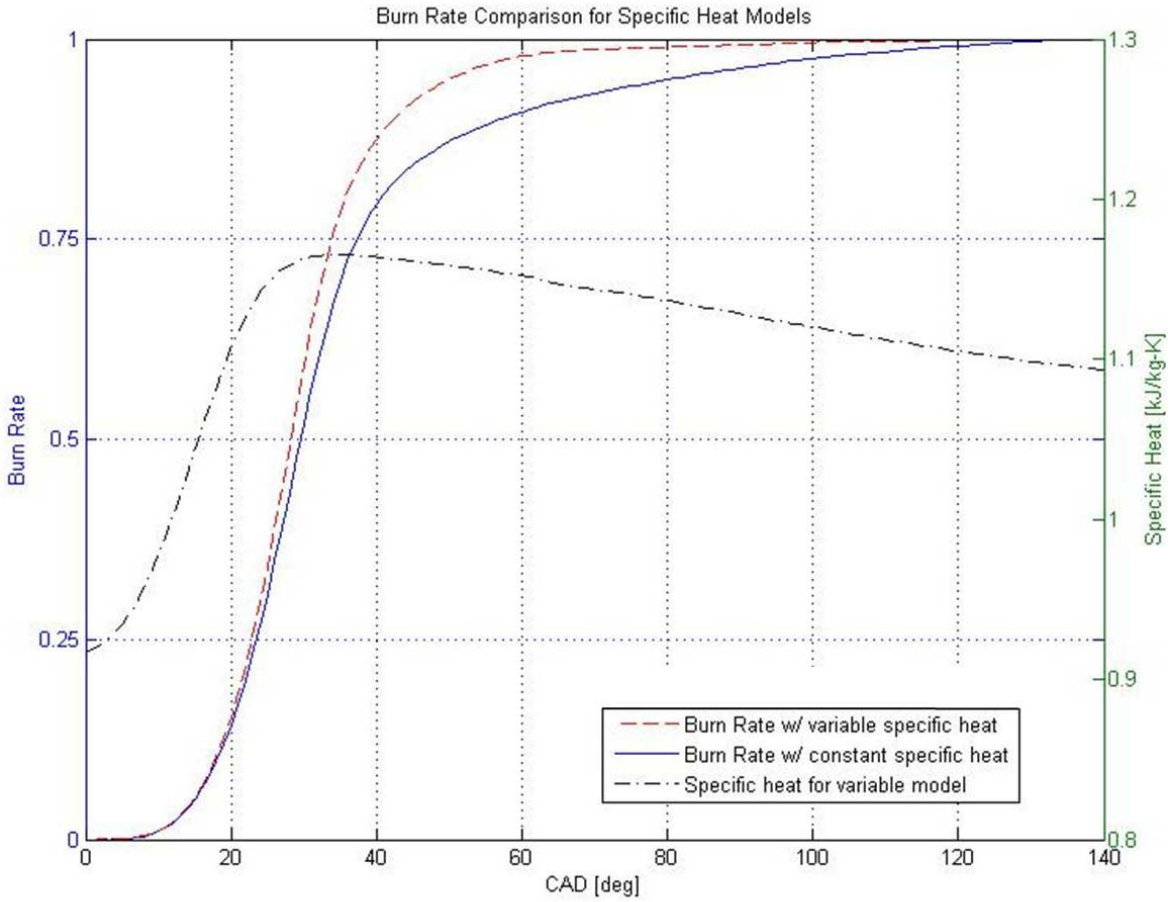


Figure 20: Burn rate comparison for constant specific heat model and variable specific heat model.

The derivation for the variable specific heat model used in this project was the same as described above up to equation 16. The derivation changes at this point because the gas constant was dependant on time since the specific heat was variable. Instead of changing equation 16 to be a function of pressure it was left in terms of temperature taking the following form with rearrangement.

$$\dot{Q}_{chemical} = m_{cyl} c_v \frac{dT_{cyl}}{dt} + \dot{Q}_{HT} + p_{cyl} \frac{dV_{cyl}}{dt} \tag{23}$$

Converting to the crank angle domain,

$$\frac{\dot{Q}_{chemical}}{d\theta} = m_{cyl} c_v \frac{dT_{cyl}}{d\theta} + \frac{\dot{Q}_{HT}}{d\theta} + p_{cyl} \frac{dV_{cyl}}{d\theta} \tag{24}$$

In order to find the heat release rate, the mass within the cylinder, the temperature and the specific heat was needed. The mass within the cylinder was found using the air/fuel ratio, the engine speed and the mass flow rate calculated from the LFE. The mass flow rate of fuel was found by,

$$\dot{m}_{fuel} = \frac{\dot{m}_{air}}{AFR} \tag{25}$$

Then using the engine speed the mass flow rates were converted to the mass within one cylinder for one cycle with,

$$m_{cyl} = \frac{120\left(\dot{m}_{air} + \dot{m}_{fuel}\right)}{4 \cdot N} \tag{26}$$

The ideal gas law was used to find the temperature within the cylinder over the cycle,

$$T = \frac{P_{cyl} V_{cyl}}{m_{cyl} R} \tag{27}$$

where P_{cyl} was the experimental cylinder pressure, V_{cyl} was the cylinder volume and m_{cyl} was the mass within the cylinder. The mass within the cylinder was determined using the mass air flow (MAF) and the air fuel ratio (AFR), thus the reason for measuring these two quantities and was constant for each cycle since it was a closed system. The pressure, volume, and gas constant of the mixture were all variables based on the crank angle thus meaning they changed over the cycle. The cylinder pressure was measured and the cylinder volume was calculated based on the crank angle using equation13.

The gas constant was dependant on the ratio of air, fuel and exhaust present in the cylinder,

$$R_{cyl} = R_{air} X_{air} + R_{fuel} X_{fuel} + R_{exhuast} X_{exhaust} \tag{28}$$

and during combustion this ratio changes as fuel and air combust to form exhaust. The ratio of air, fuel and exhaust was determined using the known initial mass and the burned mass fraction of the composition, which is the burn rate. An iteration was needed to since the burned mass changed as a function of crank angle as combustion occurred. This will be explained further in the short coming.

The gas constant for each of these three species was found using the specific heats, equation 19. Even though the specific heats are variable with temperature it was found that the difference between the two specific heats remained constant and therefore the gas constant did not depend on temperature (Moran and Shapiro). With this knowledge the gas constants for each of the species were found using the curve fit

coefficients from the lookup tables for thermodynamic properties and the following equation

$$\frac{c_i}{R} = a_{i1} + a_{i2}T + a_{i3}T^2 + a_{i4}T^3 + a_{i5}T^4 \tag{29}$$

using the inducted air temperature, thus the reason for measuring the intake air temperature, since the temperature essentially did not matter. The curve fit coefficients are specific to each different species in equation 29.

The last two unknown quantities to solve for the heat release of equation 17 where the varying specific heat and the heat transfer coefficient. With the gas constant known the temperature within the cylinder could be calculated using the ideal gas law from equation 27. This temperature was then used to find the varying specific heat for each of the three species, air, fuel and exhaust, with the curve fit coefficient equation 29. The total specific heat within the cylinder was then determined using the mass fractions,

$$c_{v_{cyl}} = c_{v_{air}} X_{air} + c_{v_{fuel}} X_{fuel} + c_{v_{exhuast}} X_{exhaust} \tag{30}$$

the same as the gas constant was calculated in equation 28. So the specific heat was essentially a function of both temperature and composition.

The heat that was lost through the cylinder walls was based on convective heat transfers,

$$\frac{dQ_{HT}}{d\theta} = Ah_c\left(T - T_w\right) \tag{31}$$

where A was the chamber surface area, T was the mean gas temperature, T_w was the mean wall temperature, and h_c is the convective heat transfer coefficient. The mean gas temperature was the temperature within the cylinder calculated with the ideal gas law. The wall temperature was determined as a combination of the coolant temperature and the temperature within the cylinder, thus the reason for needing these measurements for the combustion model. The heat transfer coefficient was found using the Woschni correlation (Woschni).

The burn rate is the cumulative heat released, therefore to find the burn rate it is just the integral of the heat release rate from equation 24.

$$x_b = \int \dot{Q}_{chemical}\, d\theta \tag{32}$$

In order to get the burned mass fraction, which essentially normalizes the function and becomes a measure of composition from 0 to 1 as explained in the chapter introduction, the burn rate has to be divided by the mass of fuel in the cylinder during combustion and the lower heating value of the fuel.

Now that it is understood how the burned mass fraction was found the iteration that was needed for the variable specific heat model can be explained. This iteration can be seen in Figure 21 and occurs at every CAD since the cylinder pressure was collected at every CAD. The gas constant and specific heat were both functions of composition

therefore the burned mass fraction was needed to calculate the composition. However the gas constant and specific heat were needed to find the burned mass fraction. So the iteration works as follows: The gas constant was determined using the composition, then the cylinder temperature was found using the ideal gas law from equation 27, followed by the species specific heats found using the curve fit coefficients and the previously found temperature, then the composition was used again to find the total specific heat. The burn mass fraction could then be found, which was then fed back into the iteration.

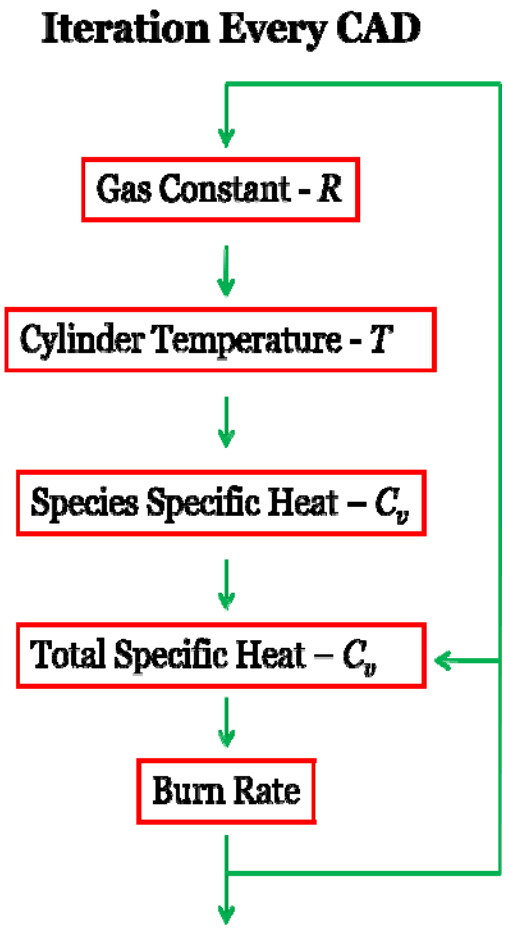


Figure 21: Iteration diagram for variable specific heat model

The variable specific heat model previously described can be simplified by assuming that the specific heat only depends on temperature instead of temperature and composition. This would allow for the burn rate to be found without using an iteration because the composition would no longer be needed in the burn rate calculation for the gas constant and the total specific heat. The gas constant would be constant and the

temperature could be found over the cycle with the ideal gas law, using the conditioned cylinder pressure and calculated cylinder volume. As can be seen in Figure 19 the temperature has a larger influences on the specific heats than the composition so this model can still be valid depending of the application of the model.

When determining the burn rate the cycle was isolated from spark timing to exhaust valve opening. The burn rate is a measure of combustion therefore before combustion is initiated the burn rate is zero and in a SI engine the spark initiates the combustion. Also the model used for burn rate assumes that the system is a closed system so once the exhaust valve opens the model was no longer valid. However by the time the exhaust valve opens the combustion is usually completed. This is why it was important to know the spark timing and exhaust valve timing.

5.3 Summary

Three variations of the inverse thermodynamic model were derived to transform the conditioned cylinder pressure into the experimental burn rate. The first model assumed constant specific heat and was the least accurate model; in the second model the specific heat was dependant on temperature and composition and was the most accurate model. The model in-between these two models assumed that the specific heat only depended on temperature. For this application the specific heat model that was dependant on both temperature and composition was implemented to find the experimental burn rate from the conditioned cylinder pressure to give the most accuracy.

This model was successful in transforming the conditioned cylinder pressure into experimental burn rates to allow for curve fits in the following section.

CHAPTER 6

SPARK IGNITION (SI) MODELING

For control applications of SI engines empirical combustion models are often used (Heywood). Two of the most common empirical models are based on either the single or double Wiebe functions. These functions can be fit to the experimental burn rate which was calculated from the cylinder pressures in the previous chapter. Appendix L shows the Matlab tool which was developed for fitting the single and double Wiebe functions. The single Wiebe function is able to accurately characterize SI combustion when there are low residuals in the combustion mixture. Significant amount of residuals in the combustion mixture slows the combustion flame front thus lengthening the combustion duration. Due to the nature of single Wiebe function, it is unable to accurately fit this type of profile. However, the double Wiebe function is capable of characterizing long combustion duration and therefore can be used when high residuals are present in the mixture. Since the engine being modeled was equipped with VVT large amounts of residuals could be present during combustion at operating conditions of large valve overlap. For this reason the double Wiebe function was used to characterize the combustion process. The following section will explain the single and double Wiebe

functions and show examples of why the double Wiebe function was chosen to model the combustion process.

When using either the single or double Wiebe function for the empirical combustion model the same basic approach is taken. A Wiebe function is fit to the experimental burn rate at every operating condition using least squares regression to find the unknown parameters (Gilat and Subramaniam). Once the unknown parameters are found for all the operating conditions, they are mapped to operating parameters of the engine, like ICAM, ECAM, AFR, spark timing, ECT. This allows the Wiebe function parameters to be determined from the engine operating conditions, which are inputs to an engine model, and thus is the basis for the combustion model. The model burn rate is thus determined by plugging the Wiebe function parameters, based on the operating condition maps, into the Wiebe function. This allows the burn rate to be determined for all operating conditions, not only conditions that were experimentally tested. The burn rate can then be used to find the cylinder pressures working backwards from the derivation in Chapter 5.

6.1 Single Wiebe Function

The single Wiebe function, which models the mass fractioned burned vs. the crank angle, takes the following form (Heywood).

$$x_b = 1 - \exp\left[-a\left(\frac{\theta - \theta_0}{\Delta\theta} \right)^{m+1} \right] \tag{33}$$

where θ is the crank angle, θ_0 is the start of combustion, $\Delta\theta$ is the total combustion duration ($x_b = 0$ to $x_b = 1$), and a and m are shape factors. The advantage of using the single Wiebe function is that only these two shape factors need to be optimized when fitting the Wiebe function to the experimental burn rates. This also means that only two parameters need to be mapped to the control inputs. The start of combustion can be modeled as the spark timing since it initiates the flame and the combustion duration can be found from the experimental burn rate. This allows two out of the four Wiebe function parameters to be based on physical characteristics of combustion.

Figure 22 shows the experimental and modeled mass fraction burned for an operating condition where there is the minimal valve overlap, 2 CAD. The modeled burn rate fits the experimental burn rate almost exactly; however since the valve overlap is minimal the residuals were also minimal. Figure 23 on the other hand shows the experimental and modeled burn mass fraction for an operating condition with a valve overlap of 52 degrees. Due to the large amount of residuals the single Wiebe function was unable to characterize the slower combustion.

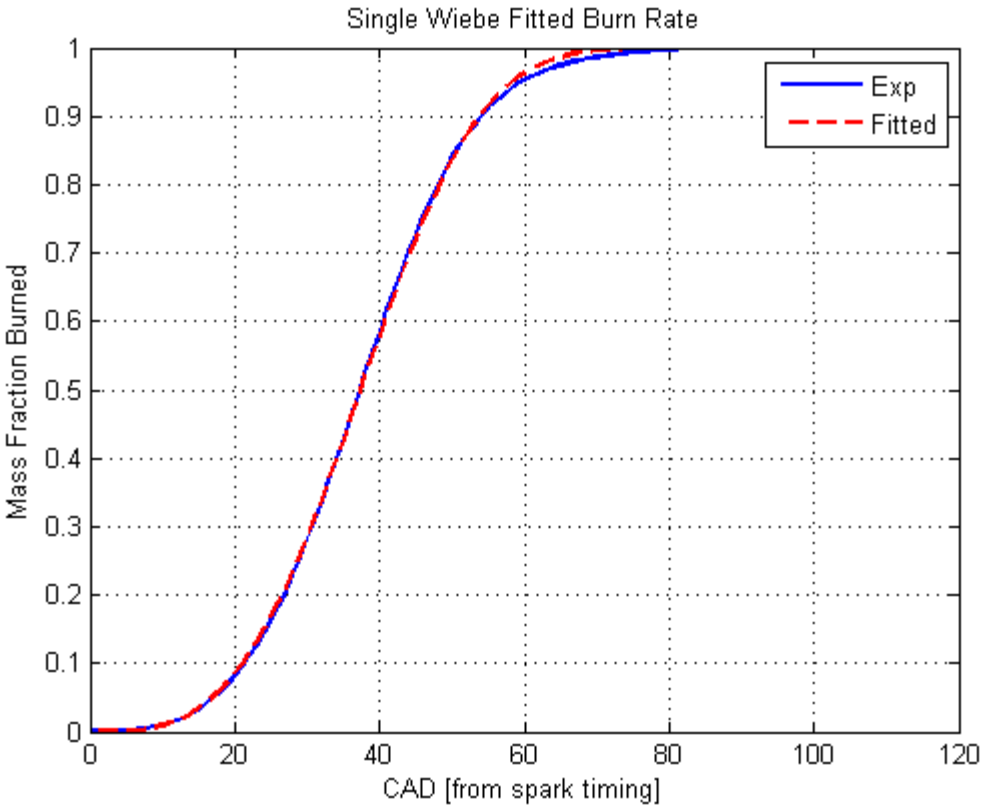


Figure 22: Experimental burn rate with single Wiebe function fit for an operating condition with 2 degrees of valve overlap.

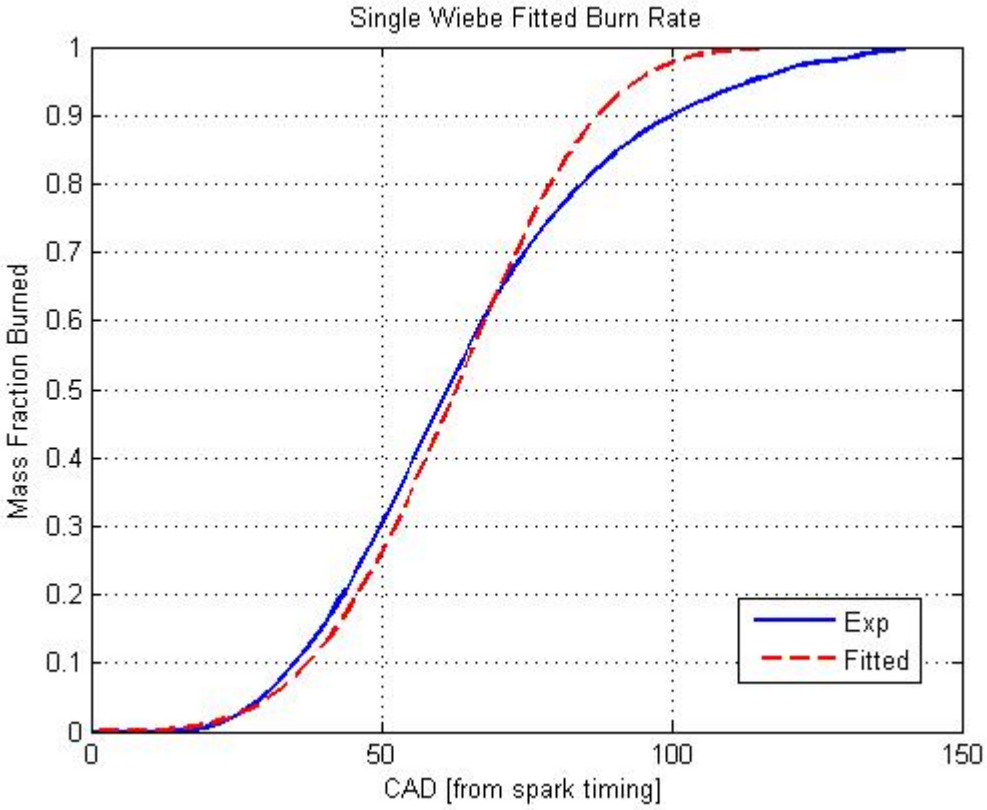


Figure 23: Experimental burn rate with single Wiebe function fit for an operating condition with 52 degrees of valve overlap.

6.2 Double Wiebe Function

The double Wiebe function, which is also a measure of the mass fraction burned vs. the crank angle, takes the following form:

$$x_b = \left\{1 - \exp\left[-a_1\left(\frac{\theta - \theta_1}{\Delta\theta_1}\right)^{m_1+1}\right]\right\} + \alpha\left\{1 - \exp\left[-a_2\left(\frac{\theta - \theta_2}{\Delta\theta_2}\right)^{m_2+1}\right]\right\} \tag{34}$$

where θ is the crank angle, θ_1 and θ_2 are the start of combustion, and $a_1, \Delta\theta_1, m_1, \alpha, a_2, \Delta\theta_2$, and m_2 are all shape factors. The double Wiebe function has more parameters than the single Wiebe function however they no longer correlate as well with physical parameters. This is the disadvantage of using the double Wiebe function because now seven parameters have to be regressed and mapped instead of two that need regressed and mapped for the single Wiebe function. When using the double Wiebe function careful attention must be focused on the regression and mapping techniques in order to minimize the error that would be associated with these processes.

Figure 24 shows the experimental and modeled burn rate for the same operating condition as shown in Figure 22 for the single Wiebe function, with low valve overlap. There is little difference between the single and double Wiebe fits. However, Figure 25 shows the double Wiebe fit for the same operating condition as that shown in Figure 23, for the single Wiebe function at large valve overlap, and there is a significant improvement in the burn rate fit. This is why the double Wiebe function was used over the single Wiebe function because it is able to accurately model the burn rate at high and low residual mixtures.

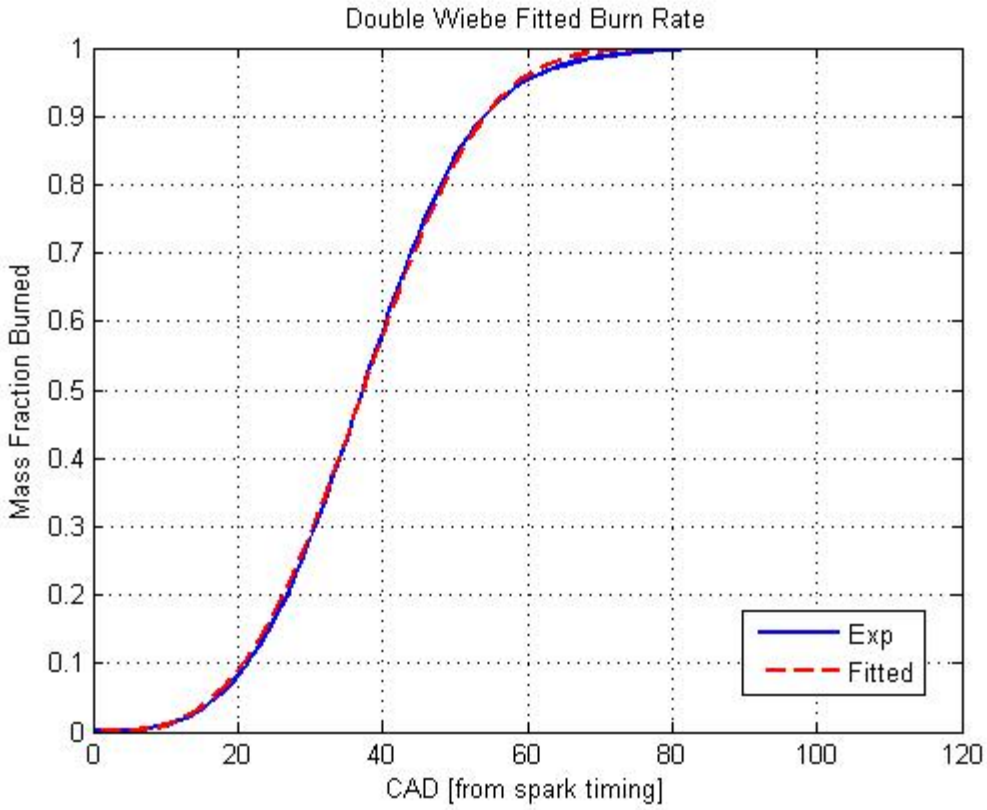


Figure 24: Experimental burn rate with double Wiebe function fit at 2 degrees of valve overlap, this is the same operating condition as in Figure 22 with the single Wiebe fit.

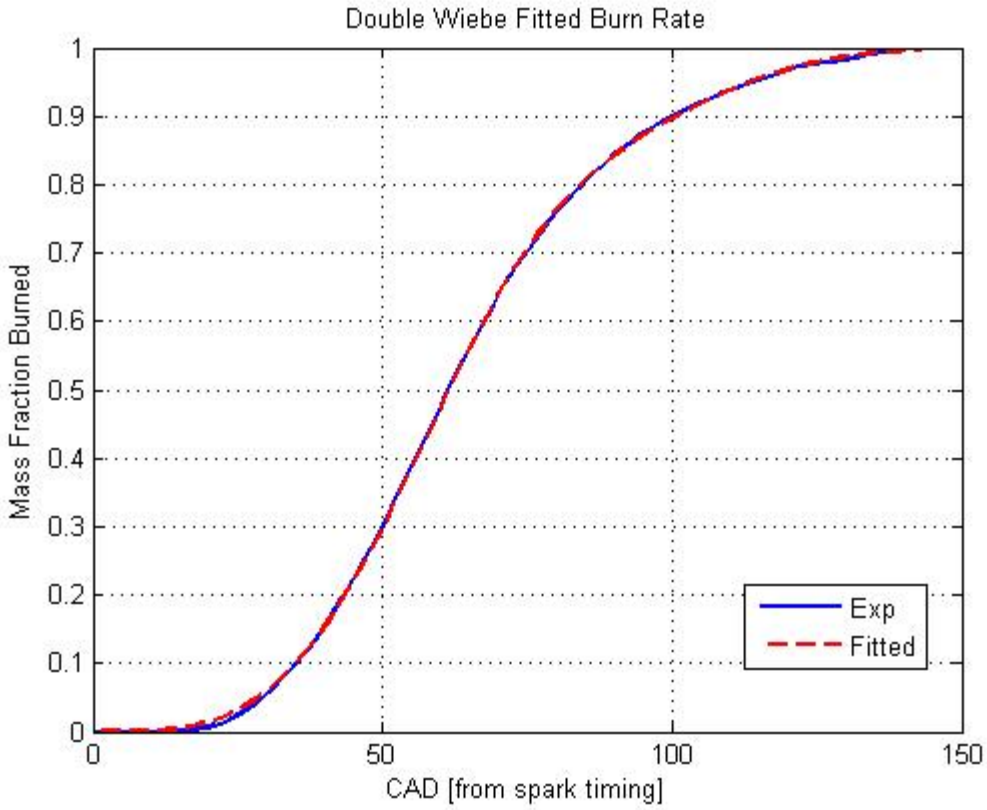


Figure 25: Experimental burn rate with double Wiebe function fit at 52 degrees of valve overlap, this is the same operating condition as in Figure 23 with the single Wiebe fit.

6.3 Summary

The single and double Wiebe functions are curve fit functions for the experimental burn rate. Using nonlinear regressions the single and double Wiebe function parameters were regressed to get accurate burn rate curve fits. The single Wiebe function parameters have more physical meaning than the double Wiebe function parameters, however the single Wiebe function was unable to accurately fit operating

conditions where large amounts of residuals were present in the combustion. Since the entire operating range of the engine was to be considered the double Wiebe function was used as the burn rate curve fitting tool.

CHAPTER 7

FUTURE WORK AND CONCLUSION

To conclude the combustion model methodology and tools, the double Wiebe function parameters need to be related to the model inputs. The mapping plays a vital role in the combustion model accuracy, so extensive research is needed to ensure that a regression is used that accurately captures the trends of the Wiebe parameters based on the control inputs. An investigation could be conducted on the individual model inputs to see their affects on the Wiebe function parameters to develop and understand the mapping process. With the finalized relationships, the Wiebe function parameters can be found from the model inputs and the burn rate profiles can be determined for the corresponding operating condition. However due to time constraints of the project this last step was not completed and was reserved for future work.

In conclusion, using well established methods for control oriented combustion modeling and cylinder pressure processing techniques, a systematic methodology was developed to transform raw experimental data into modeled combustion parameters. These combustion parameters were based on the double Wiebe function which was shown to accurately model the combustion of an advanced IC engine with variable valve timing. The methodology was developed into a set of tools using Matlab. A tool was

developed for each stage of the methodology, post-processing of the cylinder pressure, determining the experimental burn rate using the inverse thermodynamic model, and fitting the Wiebe function to the experimental burn rate. With the use of these tools the user interaction is limited to choosing the data to be processed. The automation of these tools will allow for faster control oriented combustion model calibration. The calibrated combustion model will be implemented into a larger project which models a complete advanced IC engine. The complete model will be used for control design in efforts to reduce the calibration time for new engines entering the market.

My plans for the future consist of attending graduate school at the Ohio State University to obtain my M.S. in Mechanical Engineering. I want to continue my work with engine modeling to get a more in depth understanding of the engine processes. I believe that a M.S will allow me to have a more thorough understanding in a specific area of Mechanical Engineering which I will be able to apply to the work field upon graduation.

BIBLIOGRAPHY

Brunt, M. F. and C. R. Pond. "Evaluation of Burn Rate Routines and Analysis Errors." SAE 970037 (n.d.).

Dawson, Jonathan Adam. "An Experimental and Computational Study of Internal Combustion Engine Modeling for Controls Oriented Research." Dissertation. 1998.

Ferguson, Colin R. and Allan T. Kirkpatrick. Internal Combustion Engines. New York: John Wiley and Sons Inc., 2001.

Figliola, Richard S. and Donald E. Beasley. Theory and Design for Mechanical Measurements. John Wiley and Sons, 2006.

Gatowski, J. A., et al. "Heat Release Analysis of Engine Pressure Data." SAE 841359 (1984).

Gilat, Amos and Vish Subramaniam. Numerical Methods An Introduction with Applications using Matlab. John Wiley and Sons, 2006.

Heywood, John B. Internal Combustion Engine Fundamentals. Mcgraw-Hill Inc., 1988.

Krieger, R. B. and G. L. Borman. "The Computation of Apparent Heat Release for Internal Combustion Engines." ASME 66-WA/DGP-4 (1966).

Lee, Kangyoo, Maru Yoon and Myoungho Sunwoo. "A Study on Pegging Methods for Noisy Cylinder Pressure Signal." (2007).

Moran, Michael J. and Howard N. Shapiro. Fundamentals of Engineering Thermodynamics. John Wiley and Sons, 2004.

Ponti, Fabrizio, Gabriele Serra and Carlo Siviero. "A Phenomenological Combustion Model for Common Rail Multi-Jet Diesel Engine." (2004).

Randolph, A. "Methods of Processing Cylinder-Pressure Trasnducer Signals to Maximize Data Accuracy." SAE 900170 (1990).

Woschni, G. "A Universally Applicable Equation for the Instantaneous Heat Transfer Coefficient in the Internal Combustion Engine." SAE 67093 (1967).

Yoon, Maru, Kanyoon Lee and Myougho Sunwoo. "A Method for Combustion Phasing Control using Cylinder Pressure Measurements in a CRDI Diesel Engine." (2007).

Zeng, Pin and Dennis Assanis. "Cylinder Pressure Reconstruction and its Application to Heat Transfer Analysis." SAE 010922 (2004).

Zhao, Hua and Nicos Ladommatos. Engine Combustion Instrumentation and Diagnostics. Warrendale: Society of Automotive Engineers, 2001.

APPENDIX

Appendix A: Engine Specifications

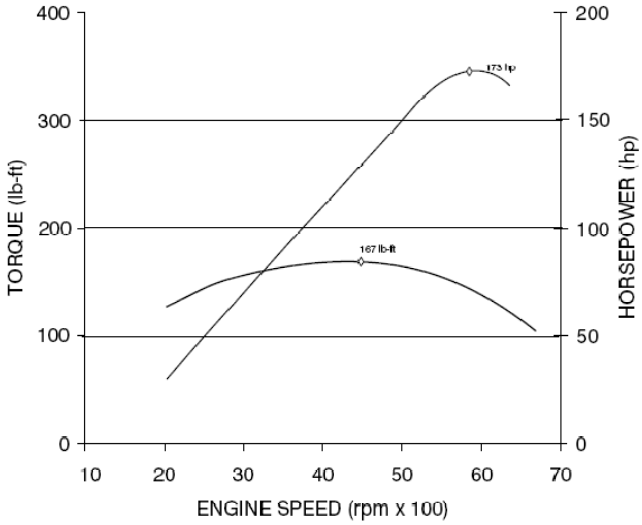
Specifications

Type: Ecotec 2.4L I4 VVT
Displacement: 146 cid (2393 cc)
Engine Orientation: Longitudinal/Transverse
Compression Ratio: 10.4:1
Valve Configuration: Dual overhead camshafts
Assembly Site: Spring Hill, TN
Valve Lifters: Hydraulic roller finger follower
Firing Order: 1 - 3 - 4 - 2
Bore x Stroke: 88 x 98 mm
Bore Center: 95.48 mm
Bore Area: 243 cm^2
Fuel System: Sequential fuel injection
Fuel Type: Regular unleaded
Horsepower:
173 hp (129 kW) @ 5800 rpm (preliminary)
Torque:
167 lb-ft (226 Nm) @ 4500 rpm (preliminary)

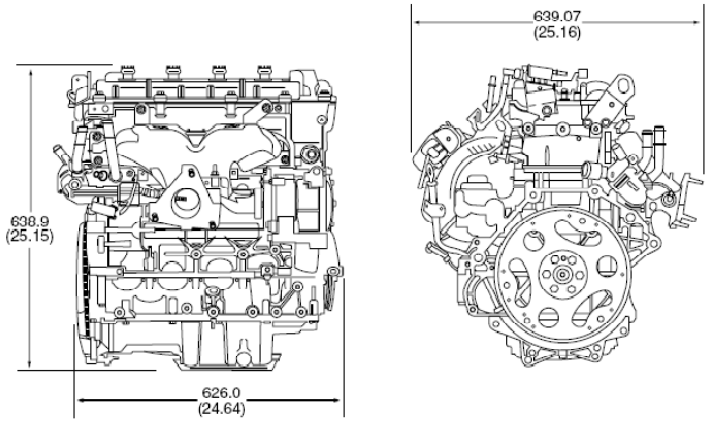
Actual power levels may vary depending on OEM calibration and application.

Fuel Shutoff: OEM defined
Shipping Weight: 273 lb (124 Kg)
Materials:
Block: Cast Aluminum
Cylinder head: Cast Aluminum
Intake manifold: Composite
Exhaust manifold: High Silicon Molybdenum, Cast Nodular Iron
Main bearing caps: Aluminum Bedplate
Crankshaft: Cast Nodular Iron
Camshaft: Cast Nodular Iron
Connecting rods: Forged Steel

Information may vary with application. All specifications listed are based on the latest product information available at the time of publication. The right is reserved to make changes at any time without notice.



Power numbers represent Automotive Pontiac Solstice application.
OEM application power numbers may vary.



GM Powertrain

Appendix B: Slow Frequency DAQ System Specifications

SCXI Thermocouple Input Modules

Specifications

Complete Accuracy Table

			Percent of Reading[2]			System Noise (peak, 3 sigma)[1]				Temperature Drift	
						Single Point		Average			
Module	Nominal Range[2]	Overall Gain[1]	Typical	Maximum	Offset (µV)	4 Hz (µV)	10 kHz or FB W (µV)	4 Hz (µV)	10 kHz or FB W (µV)	Percent of Reading/°C	Offset (µV/°C)
SCXI-1100	±10 V	1	0.03	0.05	250	600	600	15.0	15.0	0.002	24.0
	±5 V	2	0.1	0.15	175	400	400	15.0	15.0	0.002	14.0
	±2 V	5	0.1	0.15	100	200	200	15.0	15.0	0.002	8.0
	±1 V	10	0.1	0.15	75	100	100	10.0	11.0	0.002	6.0
	±500 mV	20	0.1	0.15	65	50	50	10.0	11.0	0.002	5.0
	±200 mV	50	0.1	0.15	50	30	30	10.0	11.0	0.002	4.5
	±100 mV	100	0.1	0.15	50	10	10	5.0	6.0	0.002	4.5
	±50 mV	200	0.1	0.15	50	10	10	5.0	6.0	0.002	4.5
	±20 mV	500	0.1	0.15	45	10	10	5.0	6.0	0.002	4.0
	±10 mV	1000	0.1	0.15	45	1.5	1.5	0.5	1.5	0.002	4.0
	±5 mV	2000	0.1	0.15	45	1.5	1.5	0.5	1.5	0.002	4.0

[1]Absolute Accuracy (15 to 35 °C). To calculate the absolute accuracy for the SCXI-1100, 1102, 1102B, 1102C, 1104, 1104C, and/or 1112, visit ni.com/accuracy.

			Percent of Reading[1]			System Noise (peak, 3 sigma)[1]		Temperature Drift	
Module	Nominal Range[2]	Overall Gain[1]	Typical	Maximum	Offset (µV)	Single Point	Average (µV)	Percent of Reading/°C	Offset (µV/°C)
SCXI-1102	±10 V	1	0.025	0.035	500	600 µV	50	0.0010	20
	±100 mV	100	0.015	0.02	25	20 µV	5	0.0005	1
SCXI-1102B	±10 V	1	0.025	0.035	500	600 µV	50	0.0010	20
	±100 mV	100	0.015	0.02	25	20 µV	5	0.0005	1
SCXI-1102C	±10 V	1	0.025	0.035	500	600 µV	70	0.0010	20
	±100 mV	100	0.015	0.02	25	30 µV	10	0.0005	1
SCXI-1104	±60 VDC	0.1	0.035	0.05	800	900 µV	200	0.0020	50
SCXI-1104C	±42 VAC	0.1	0.035	0.05	800	1 mV	300	0.0020	50
SCXI-1112	±100 mV	100	0.015	0.02	25	20 µV	5	0.0005	1

[1]Absolute Accuracy (15 to 35 °C). To calculate the absolute accuracy for the SCXI-1100, 1102, 1102B, 1102C, 1104, 1104C, and/or 1112, visit ni.com/accuracy.

Input Characteristics

Module	Number of Channels
SCXI-1100, SCXI-1102, SCXI-1102B, SCXI-1102C, SCXI-1104, SCXI-1104C	32 differential
SCXI-1112	8 differential

Input signal ranges ..... See accuracy table
Input coupling ..... DC

Maximum working voltage

Module	Maximum Working Voltage (Signal + Common Mode)
SCXI-1100, SCXI-1102, SCXI-1102B, SCXI-1102C	±10 V
SCXI-1104, SCXI-1104C	30 V_{rms} or ±42 VAC peak or 60 VDC
SCXI-1112	±10 V

Overvoltage protection

Module	Powered On	Powered Off
SCXI-1100	±25 V	±15 V
SCXI-1102, SCXI-1102B, SCXI-1102C	±42 V	±42 V
SCXI-1104, SCXI-1104C	30 V_{rms} or ±42 VAC peak or 60 VDC	
SCXI-1112	±42 V	±42 V

Inputs with Overvoltage Protection	
SCXI-1100, SCXI-1104, SCXI-1104C	CH0..CH31
SCXI-1102, SCXI-1102B, SCXI-1102C	CH0..CH31, CJ Sensor
SCXI-1112	CH0..CH7, CJ Sensor

Offset error ..... See accuracy table
Gain error ..... See accuracy table

Transfer Characteristics

Nonlinearity

Module	Percentage of Full Scale Range
SCXI-1100	±0.006
SCXI-1102, SCXI-1102B, SCXI-1102C	±0.005
SCXI-1104, SCXI-1104C	±0.01
SCXI-1112	±0.005

Amplifier Characteristics

Input impedance

Module	Normal Powered On	Powered Off/Overload
SCXI-1100	>1 GΩ	1.6 kΩ
SCXI-1102, SCXI-1102B, SCXI-1102C	>1 GΩ	10 kΩ
SCXI-1104, SCXI-1104C	1 MΩ	800 kΩ
SCXI-1112	>1 GΩ	10 kΩ

SCXI Thermocouple Input Modules

Specifications

Input bias current

Module	Current
SCXI-1100	±350 pA
SCXI-1102, SCXI-1102B, SCXI-1102C, SCXI-1104, SCXI-1104C, SCXI-1112	±0.5 nA

Input offset current

Module	Current
SCXI-1100	±350 pA
SCXI-1102, SCXI-1102B, SCXI-1102C, SCXI-1104, SCXI-1104C, SCXI-1112	±1.0 nA

No MMR

CMRR (Common Mode Rejection Ratio) (DC to 60 Hz)

Module	Range	Filter	CMRR (dB)
SCXI-1100	±10 V to ±2 V	4 Hz	98
	±1 V to ±5 mV	4 Hz	78
	±10 V to ±2 V	10 kHz or full bandwidth	110
	±1 V to ±5 mV	10 kHz or full bandwidth	90
SCXI-1102	±10 V to ±100 mV	2 Hz	110
SCXI-1102B	±10 V to ±100 mV	200 Hz	90
SCXI-1102C	±10 V to ±100 mV	10 kHz	90
SCXI-1104	±42 VAC/±60 VDC	2 Hz	70
SCXI-1104C	±42 VAC/±60 VDC	10 kHz	70
SCXI-1112	±100 mV	2 Hz	110

Output range ..... See accuracy table

No output impedance

Dynamic Characteristics

Input signal bandwidth

Module	Bandwidth
SCXI-1100	4 Hz, 10 kHz, full bandwidth
SCXI-1102	2 Hz
SCXI-1102B	200 Hz
SCXI-1102C	10 kHz
SCXI-1104	2 Hz
SCXI-1104C	10 kHz
SCXI-1112	2 Hz

Step response (10 V step)

Module	Filter Setting	Range	Accuracy ±0.012%[1]	Accuracy ±0.008%[2]	Accuracy ±0.0015%[2]
SCXI-1100	Full bandwidth	±10 V to ±100 mV	6 µs	10 µs	32 µs
	(No filter)	±50 mV	7.5 µs	10 µs	33 µs
		±20 mV	12 µs	25 µs	40 µs
		±10 mV	20 µs	26 µs	76 µs
		±5 mV	25 µs	30 µs	105 µs
	10 kHz	All ranges	150 µs	160 µs	200 µs
	4 Hz	All ranges	350 ms	400 ms	500 ms

Module	Filter Setting	Range	Accuracy ±0.01%	Accuracy ±0.01%
SCXI-1102	2 Hz	±10 V, ±100 mV	1 s	10 s
SCXI-1102B	200 Hz	±10 V, ±100 mV	10 ms	100 ms
SCXI-1102C	10 kHz	±10 V, ±100 mV	200 µs	1 ms
SCXI-1112	2 Hz	±100 mV	1 s	10 s

Module	Filter Setting	Range	Accuracy ±0.01%	Accuracy ±0.01%
SCXI-1104	2 Hz	±42 VAC/±60 VDC	1 s	10 s
SCXI-1104C	10 kHz	±42 VAC/±60 VDC	200 µs	1 ms

Multiplexer performance

Module	Scan Interval Settle to ±0.012%[1]	Scan Interval Settle to ±0.008%[2]
SCXI-1100	(See step response)	(See step response)
SCXI-1102, SCXI-1102B, SCXI-1102C, SCXI-1104, SCXI-1104C, SCXI-1112	3 µs	10 µs

Filter Characteristics

Module	Filter Setting	Range	Noise Related to Input (pV_{rms})
SCXI-1100	Full bandwidth	±10 V	15
	(No filter)	±10 mV	6
	4 Hz	±10 V	15
		±10 mV	0.2
	10 kHz	±10 V	15
		±10 mV	1.5
SCXI-1102	2 Hz	±100 mV	5
		±10 V	50
SCXI-1102B	200 Hz	±100 mV	5
		±10 V	50
SCXI-1102C	10 kHz	±10 V	70
SCXI-1104	2 Hz	±100 mV	10
		±42 VAC, ±60 VDC	200
SCXI-1104C	10 kHz	±42 VAC, ±60 VDC	300
SCXI-1112	2 Hz	±100 mV	5

Type ..... RC

Cutoff Frequency (-3 dB)

SCXI-1100 ..... 4 Hz, 10 kHz, full bandwidth (jumper-selectable)

SCXI-1102, 1104, 1112 ..... 2 Hz

SCXI-1102B ..... 200 Hz

SCXI-1102C, 1104C ..... 10 kHz

Stability

Module	Input Range	Gain Temperature Coefficient (ppm/°C)	Offset Temperature Coefficient (µV/°C)
SCXI-1100	All ranges	20	20
SCXI-1102	±10 V	10	20
SCXI-1102B SCXI-1102C	±100 mV	10	1
SCXI-1104 SCXI-1104C	±42 VAC, ±60 VDC	20	50
SCXI-1112	±100 mV	10	1

Recommended warm-up time ..... 20 minutes

Physical

Dimensions ..... 9.0 by 17.3 by 30.3 cm
1.2 by 6.8 by 8.0 in.

I/O Connector

Rear ..... 50-pin male ribbon cable rear connector

Front

SCXI-1112 ..... 8 uncompensated mini thermocouple connectors

Others ..... 96-pin male DIN C front connector

Environment

Operating temperature ..... 0 to 50 °C

Storage temperature ..... -55 to 150 °C

Relative humidity ..... 5 to 90% noncondensing

Certification and Compliance

European Compliance

EMC ..... EN 61326 Group I Class A, 10 m, Table 1 Immunity

Safety ..... EN 61010-1

North American Compliance

EMC ..... FCC Part 15 Class A using CISPR

Australia and New Zealand Compliance

EMC ..... AS/NZS 2064.1/2 (CISPR-11)

[1] Includes effects of NI 6052E with 1 or 2 m SCXI cable assembly.
[2] Includes effects of NI 6030E with 1 or 2 m SCXI cable assembly.

For a definition of specific terms, please visit ni.com/glossary.

Appendix C: Differential Pressure Sensor and LFE Data Sheet

Model FP2000

PERFORMANCE SPECIFICATIONS

Characteristic	Measure
Accuracy [1]	See accuracy table
Output (selectable)	mV/V (see accuracy table), 0 Vdc to 5 Vdc, 0 Vdc to 10 Vdc, or 4 mA to 20 mA (two wire)
Resolution	Infinite

ENVIRONMENTAL SPECIFICATIONS

Characteristic	Measure
Temperature, operating	-40 °C to 116 °C [-40 °F to 240 °F]
Temperature, compensated	4 °C to 60 °C [40 °F to 140 °F] [2]
Temperature, error band [2]	
0.10 % accuracy	±0.5 % full scale
0.25 % accuracy	±1.0 % full scale

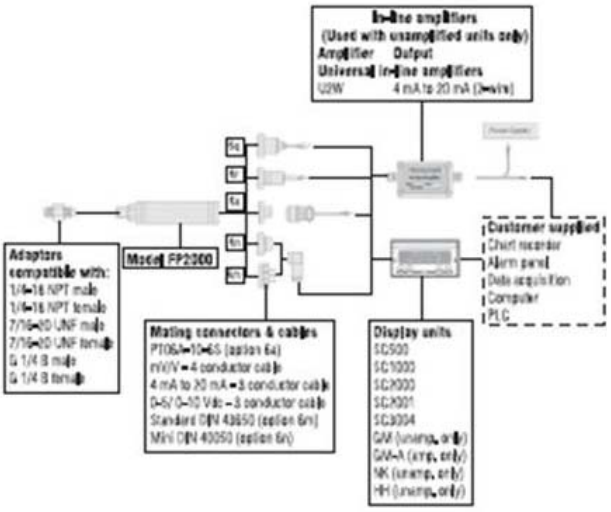
ELECTRICAL SPECIFICATIONS

Characteristic	Measure
Excitation (calibration)	
Amplified	
(4 mA to 20 mA; 0 Vdc to 5 Vdc)	9 Vdc to 28 Vdc
Amplified	
(0 Vdc to 10 Vdc)	15 Vdc to 28 Vdc
Unamplified (mV/V)	10 Vdc

MECHANICAL SPECIFICATIONS

Characteristic	Measure
Media 3	Gas, liquid
Overload - safe	
1000 psi and below	4X full scale or 3000 psi, whichever is less
1500 psi and above	4X full scale or 15000 psi, whichever is less
Negative direction (for differential)	4X full scale or 250 psi, whichever is less
Overload - burst	
1000 psi and below	3000 psi
1500 psi and above	15000 psi
Pressure port	200 % over capacity
Wetted parts material	Ha C276 & 316L stainless steel

TYPICAL SYSTEM DIAGRAM



LFE

Cert. No.:080405071754
Filename: 70321001.1NW
Program: LAMINAR 1.47

Standard

Customer name: OHIO STATE UNIVERSITY
Customer address: ACCTING DEPT 4TH FLOOR
1960 KENNY ROAD COLUMBUS OH 43210
Customer order no.: RF01025521
Meriam order no.: 703210

LFE model no.: 50MH10-5
LFE serial no.: 703210-W1
Curve no.: 37367
Calibration date: 07-31-2005
Todays date: 08-04-2005

Plot no.: 116
Working master serial no.: WMMH10-6
Master master serial no.: MMMH10-6

uut = Unit Under Test
mas = Master

Calibration data from flow lab

RH %	Tuut DEG F	TMAS# DEG F	PSuut PSIA	PSMAS PSIA	DPuut In H2O	DPmas In H2O
47.2	70.3	72.4	14.367	14.334	1.007	0.704
46.8	70.1	72.1	14.367	14.297	2.012	1.407
46.6	70.0	71.8	14.364	14.258	3.001	2.104
47.0	69.9	71.4	14.362	14.222	3.994	2.800
47.1	69.7	71.0	14.361	14.183	5.013	3.521
47.2	69.6	70.6	14.357	14.144	6.025	4.230
47.4	69.6	70.3	14.354	14.107	6.981	4.909
47.3	69.5	70.1	14.351	14.067	8.033	5.651

Master LFE Coefficients:
A0 = 9.26338E+03 A3 = 0.00000E+00 A6 = 0.00000E+00
A1 = -2.46838E+07 A4 = 0.00000E+00 A7 = 0.00000E+00
A2 = -2.78720E+11 A5 = 0.00000E+00 A8 = 0.00000E+00

Room conditions at start (inlet read): 70.4 DEG F 14.318 PSIA 51.4 RH
Room conditions at end: (inlet read): 69.5 DEG F 14.367 PSIA 49.8 RH

Reduced data, based on master lfe coefficients:

DATA POINT	DP uut InH2O@4C	FLOW IN ACFM BASED ON MASTER	CFM* (DATA) BASED ON MASTER	CFM* (CURVE) B*DP+C*DP^2	PERCENT ERROR**
1	1.007	35.409082	35.404871	35.410899	0.02
2	2.012	70.288138	70.260672	70.336512	0.11
3	3.001	104.466548	104.415072	104.323761	-0.09
4	3.994	138.196426	138.109531	138.092093	-0.01
5	5.013	172.645385	172.495712	172.325756	-0.10
6	6.025	206.082073	205.875354	205.926210	0.02
7	6.981	237.647614	237.384957	237.299820	-0.04
8	8.033	271.475118	271.147335	271.378455	0.09

*CFM = ACFM x (Flowing viscosity in Micropoise / 181.87)
** PERCENT ERROR = ((CURVE-DATA) / DATA) * 100

A Least Squares Fit of the CFM(DATA) yields the following formula and LFE uut coefficients used to generate the CFM(CURVE) values:

CFM(CURVE) = (B x DP) + (C x DP^2) Where B = 3.53544E+01
C = -1.95440E-01

A Flow of 270.327271 CFM produces a UUT DP of 8.00 In H2O

TERMINAL NON-LINEARITY = 2.363 %
INDEPENDENT NON-LINEARITY = 0.702 %

FOR DETAILS SEE FILE NO. 501:440

Appendix D: High Frequency DAQ System Specifications

Low-Cost E Series Multifunction DAQ – 12 or 16-Bit, 200 kS/s, 16 Analog Inputs

NI E Series – Low-Cost

- 16 analog inputs at up to 200 kS/s, 12 or 16-bit resolution
- Up to 2 analog outputs at 10 kS/s, 12 or 16-bit resolution
- 8 digital I/O lines (TTL/CMOS); two 24-bit counter/timers
- Digital triggering
- 4 analog input signal ranges
- NI-DAQ driver that simplifies configuration and measurements

Families

- NI 6036E
- NI 6034E
- NI 6025E
- NI 6024E
- NI 6023E

Operating Systems

- Windows 2000/NT/XP
- Real-time performance with LabVIEW
- Others such as Linux® and Mac OS X

Recommended Software

- LabVIEW
- LabWindows/CVI
- Measurement Studio
- VI Logger

Other Compatible Software

- Visual Basic, C/C++, and C#

Driver Software (included)

- NI-DAQ 7

Family	Bus	Analog Inputs	Input Resolution	Max Sampling Rate	Input Range	Analog Outputs	Output Resolution	Output Rate	Output Range	Digital I/O	Counter/Timers	Triggers
NI 6036E	PCI, PCMCIA	16 SE/8 DI	16 bits	200 kS/s	±0.05 to ±10 V	2	16 bits	10 kS/s[1]	±10 V	8	2, 24-bit	Digital
NI 6034E	PCI	16 SE/8 DI	16 bits	200 kS/s	±0.05 to ±10 V	0	–	–	–	8	2, 24-bit	Digital
NI 6025E	PCI, PXI	16 SE/8 DI	12 bits	200 kS/s	±0.05 to ±10 V	2	12 bits	10 kS/s[1]	±10 V	8	2, 24-bit	Digital
NI 6024E	PCI, PCMCIA	16 SE/8 DI	12 bits	200 kS/s	±0.05 to ±10 V	2	12 bits	10 kS/s[1]	±10 V	8	2, 24-bit	Digital
NI 6023E	PCI	16 SE/8 DI	12 bits	200 kS/s	±0.05 to ±10 V	0	–	–	–	8	2, 24-bit	Digital

[1] 10 kS/s typical when using the single DMA channel for analog output. 1 kS/s maximum when using the single DMA channel for either analog input or counter/timer operations. 1 kS/s maximum for PCMCIA DAQCard devices in all cases.

Table 1. Low-Cost E Series Model Guide

Overview and Applications

National Instruments low-cost E Series multifunction data acquisition devices provide full functionality at a price to meet the needs of the budget-conscious user. They are ideal for applications ranging from continuous high-speed data logging to control applications to high-voltage signal or sensor measurements when used with NI signal conditioning. Synchronize the operations of multiple devices using the RTSI bus or PXI trigger bus to easily integrate other hardware such as motion control and machine vision to create an entire measurement and control system.

Highly Accurate Hardware Design

NI low-cost E Series DAQ devices include the following features and technologies:

Temperature Drift Protection Circuitry – Designed with components that minimize the effect of temperature changes on measurements to less than 0.0010% of reading/°C.

Resolution-Improvement Technologies – Carefully designed noise floor maximizes the resolution.

Onboard Self-Calibration – Precise voltage reference included for calibration and measurement accuracy. Self-calibration is completely software controlled, with no potentiometers to adjust.

NI DAQ-STC – Timing and control ASIC designed to provide more flexibility, lower power consumption, and a higher immunity to noise and jitter than off-the-shelf counter/timer chips.

NI MITE – ASIC designed to optimize data transfer for multiple simultaneous operations using bus mastering with one DMA channel, interrupts, or programmed I/O.

NI PGIA – Measurement and instrument class amplifier that guarantees settling times at all gains. Typical commercial off-the-shelf amplifier components do not meet the settling time requirements for high-gain measurement applications.

PFI Lines – Eight programmable function input (PFI) lines that you can use for software-controlled routing of interboard and intraboard digital and timing signals.

RTSI or PXI Trigger Bus – Bus used to share timing and control signals between two or more PCI or PXI devices to synchronize operations.

RSE Mode – In addition to differential and nonreferenced single-ended modes, NI low-cost E Series devices offer the referenced single-ended (RSE) mode for use with floating-signal sources in applications with channel counts higher than eight.

Onboard Temperature Sensor – Included for monitoring the operating temperature of the device to ensure that it is operating within the specified range.

NATIONAL INSTRUMENTS

32-Bit Counter/Timers

NI 660x

- Up to eight 32-bit counter/timers
- 80 MHz maximum source frequency (125 MHz with prescalers)
- Debouncing and glitch removal
- High-stability timebase (PXI-6608 only)
- GPS-based synchronization (PXI-6608 only)
- NI DAQ driver simplifies configuration and measurements

Models

- NI PCI-6601
- NI PCI-6602
- NI PXI-6602
- NI PXI-6608

Operating Systems

- Windows 2000/NT/XP
- Real-time performance with LabVIEW (p. 134)
- Others such as Linux and Mac OS X (page 187)

Recommended Software

- LabVIEW
- LabWindows/CVI
- Measurement Studio

Other Compatible Software

- Visual Basic
- Visual C/C++, C#

Driver Software (included)

- NI-DAQ 7

Calibration Certificate Included

See page 21.

Overview and Applications

NI 660x devices are timing and digital I/O (DIO) modules for PCI and PXI. They offer up to eight 32-bit counter/timers and up to 32 lines of 5 V TTL/CMOS-compatible digital I/O. You can perform a wide variety of buffered measurements or other counter/timer tasks with NI 660x devices, including position or quadrature encoder measurement, event counting, period measurement, pulse-width measurement, frequency measurement, pulse generation, and pulse-train generation.

Features

Counter/Timers

The NI 660x devices are equipped with the NI-TIO ASIC, a National Instruments counter and digital I/O ASIC for advanced timing and counting applications. Each NI 6602 and NI 6608 device features two NI-TIO ASICs to provide a total of eight counter/timers.

The PCI-6601 board features one NI-TIO ASIC for a total of four counter/timers. The counters are software-compatible with those found on E Series multifunction DAQ devices, but NI 660x devices offer additional capabilities.

Each counter has a gate, up/down, and source signal, which can be controlled by external or internal signals. Each counter has one output that can be routed externally or to other counters on the device. 20 MHz and 100 kHz timebases are available on each device for use with each counter/timer. In addition, an 80 MHz timebase is available on the NI 6602 and NI 6608 devices. A hardware trigger can be used to start multiple counters simultaneously. See Table 1 for more information.

Family	Bus	Counter/ Timers	Size	Max Source Frequency	Compatibility	Digital I/O	Pulse Generation	Buffered Operations	Debouncing/ Glitch Removal	Oscillator Stability	GPS Synchr.	Buffered Operations[2] DMA	Buffered Operations[2] Interrupt
NI 6601	PCI	4	32 bits	20 MHz[1]	5 V TTL/CMOS	Up to 32	✓	✓	✓	50 ppm	–	1	3
NI 6602	PCI PXI	8	32 bits	80 MHz[1]	5 V TTL/CMOS	Up to 32	✓	✓	✓	50 ppm	–	3	5
NI 6608	PXI	8	32 bits	80 MHz[1]	5 V TTL/CMOS	Up to 32	✓	✓	✓	75 ppb	✓	3	5

[1] Max Source Frequency with prescalers is 80 MHz for the NI 6601 and 125 MHz for the NI 6602 and NI 6608. These frequencies are dependent on drive strength of input signal and cable length. Consider these speeds to be the maximum. [2] DMA transfers have higher throughput than interrupt transfers.

Table 1. NI 660x Products Specifications Summary (See page 393 for detailed specifications.)

Appendix E: Cylinder Pressure Sensor Specifications

Standard Specifications

Measuring Range	0...250 bar (3625 psi) 25 MPa
Lifetime	> 10^8 load changes
Overload	300 bar (4350 psi) 30 MPa
Sensitivity (nominal)	15 pC/bar (1.03 pC/psi) 150 pC/MPa
Linearity	< ±0.3 (0.2)* % FSO
Natural Frequency	115 kHz
Acceleration Sensitivity	< 0.001 bar/*g*
Shock Resistance	> 2000 *g*
Operating Temperature Range	up to 400°C (750°F)
Thermal Sensitivity Shift	20...400°C < ±2%, 200...300°C < ±0.5%
Insulation Resistance at 20°C (68°F)	> 10^{13} Ω
Capacitance	7 pF
Mass (without cable)	2.3 grammes
Mounting Torque	1.5 Nm (for adaptors refer to drawings)

Thermodynamic Specifications

Cyclic Temperature Drift	< ±0.5 (0.3)* bar
Load Change Drift	
Max. Zero-line Gradient dp/dt	1 mbar/ms
Permanent Zero-line Deviation	7 bar
IMEP-Stability	< 3 %

*) Selected transducers GH12D (Art. No. GG0721) with specifications in ( ) can be supplied upon request.

Appendix F: Exhaust Pressure Sensor Specifications

Piezoresistive Absolute Pressure Sensors – Universal Precision Pressure Sensors,
Types 4043A..., 4045A..., 4073A... and 4075A...

KISTLER
measure. analyze. innovate.

Technical Data

Sensor Designs

Sensor Type		4043A...	4073A...	4045A...	4075A...
Process connection		M14x1,25	M12x1	M14x1,25	M12x1
Compensated operating temperature range	°C	–20 ... 50		20 ... 120	
Min./max. temperature	°C	–40/70		0/140	

General Technical Data for Types 4043A.../4045A...

Sensor Type		A1	A2	A5	A10	A20	A50	A100	A200	A500
Measuring range	bar	0 ... 1	0 ... 2	0 ... 5	0 ... 10	0 ... 20	0 ... 50	0 ... 100	0 ... 200	0 ... 500
Overload	bar	2,5	5	12,5	25	50	125	250	500	750
Sensitivity at Ical	mV/bar	500	250	100	50	25	10	5	2,5	1
Natural frequency	kHz	>20	>30	>80	>120	>150	>180	>200	>200	>200
Linearity (EP)	±%FSO	<0,3								
Tightening torque	N-m	12 ... 20								
Weight	g	33								

General Technical Data for Types 4073A.../4075A...

Sensor Type		A10	A20	A50	A100	A200	A500
Measuring ranges	bar	0 ... 10	0 ... 20	0 ... 50	0 ... 100	0 ... 200	0 ... 500
Overload	bar	25	50	125	250	500	750
Sensitivity at Ical	mV/bar	50	25	10	5	2,5	1
Natural frequency	kHz	>120	>150	>180	>200	>200	>200
Linearity (EP)	±%FSO	<0,3					
Tightening torque	N-m	12 ... 20					
Weight	g	28					

General Technical Data for Types 4043A.../4045A.../4073A.../4075A...

Calibration current	mA	2 ... 5
Reference current	mA	4
Input and output impedance	kΩ	≈3
Stability:		
of sensitivity	%/a	<0,2 (for 1bar ≦±0,5%)
of zero measurand output	%FSO/a	<0,5
Thermal zero shift	±%/FSO	<0,5
Thermal sensitivity shift	±%	<±1,0
Acceleration sensitivity	mbar/g	<0,3
Shock resistance	g	1 000
Protection		IP65

Materials

Diaphragm	Mat. No.	1.4435
Sensor case	Mat. No.	1.4301
Cable		Viton®

Appendix G: Crankshaft Encoder Specifications

Model H25® Incremental Encoder

BEI Industrial Encoders

The H25 is the flagship of the BEI Industrial Encoders product line. It was designed from the ground up for the industrial marketplace. The H25 offers features such as EMI shielding, 40 lb. ABEC 7 bearings, matched thermal coefficients on critical components, and custom high-efficiency optics. The encoder meets NEMA 4 and 13 requirements when ordered with the shaft seal. Typical applications include machine control, process control, the wood processing industry, oil well logging, industrial weighing, agricultural machinery, textile equipment, web process control, robotics, and food processing.

The H25 Incremental Encoder is available with the following certifications:

- EN 55011 and EN 61000-6-2
- CSA Class I, Div1 Group C&D
- CENELEC EEX ia IIC T4
- U.S. Standards Class I, Group A,B,C & D; Class II Group E, F & G
- Canadian Standards Class I, Zone 0, Group IIC

Mechanical Specifications

Shaft Diameter: 3/8" (1/2" as special feature)

Flat On Shaft: 3/8" Shaft: 0.80 long X 0.03" deep; 1/2" Shaft: 0.80 long X 0.04" deep

Shaft Loading: 3/8" shaft: Up to 40 pounds axial and 35 pounds radial; 1/2" shaft: Up to 90 pounds axial and 80 pounds radial

Shaft Runout: 0.0005 T.I.R. at midpoint regardless of shaft diameter

Starting Torque at 25°C: Without shaft seal 1.0 in-oz (max); With shaft seal 2.5 in-oz (max); 1/2" shaft with shaft seal: 3.5 in-oz (max)

Bearings: Class ABEC 7 standard, ABEC 5 for 1/2" shaft

Shaft Material: 416 stainless steel

Bearing Housing: Die cast aluminum with protective finish; stainless steel (special feature)

Cover: Die cast aluminum; stainless steel (special feature)

Bearing Life: 2 X 10^8 revs (1300 hrs at 2500 RPM) at rated load 1 X 10^{10} revs (67,000 hrs at 2500 RPM) at 10% of rated load

Maximum RPM: 12,000 RPM nominal, 8000 RPM with 1/2" shaft (see Frequency Response, below) 30,000 RPM available on units with 3/8" shaft—consult with factory

Moment of Inertia: 4.1 X 10^{-4} oz-in-sec^2; 5.2 X 10^{-4} oz-in-sec^2 with 1/2" shaft

Weight: 13 oz typical, 14.5 oz typical with 1/2" shaft

Electrical Specifications

Code: Incremental

Output Format: 2 channels in quadrature, 1/2 cycle index gated with negative B channel

Cycles Per Shaft Turn: 1 to 72,000 (see table 2) For resolutions above 3,600 see BEI for interpolation options

Supply Voltage: 5 to 28 VDC available

Current Requirements: 100 mA typical + output load, 250 mA (max)

Voltage/Output: (see note 5)
15V/V: Line Driver, 5–15 VDC In, $V_{out} = V_{in}$
28V/V: Line Driver, 5–28 VDC In, $V_{out} = V_{in}$
28V/5: Line Driver, 5–28 VDC In, $V_{out} = 5$ VDC
28V/OC: Open Collector, 5–28 VDC In, OC_{out}

Protection Level: Reverse, overvoltage and output short circuit (see note 5)

Frequency Response: 100 kHz, up to 1MHz with interpolation option (see note 7)

Output Terminations: (See table 1, back)

Note: Consult factory for other electrical options

Environmental Specifications

Enclosure Rating: NEMA 4 & 13 (IP 66) when ordered with shaft seal (on units with an MS connector) or a cable gland (on units with cable termination).

Temperature: Operating, 0° to 70° C; extended temperature testing available (see note 8); Storage, -25° to 90° C unless extended temperature option called out.

Shock: 50 g's for 11 msec duration

Vibration: 5 to 2000 Hz @ 20 g's

Humidity: 98% RH without condensation

NOTES & TABLES: All notes and tables referred to in the text can be found on the back of this page.

Appendix H: Cylinder Pressure Post-Processing Interface

```
%GM Modeling Data Processing
%**********************************************************************
****
%
%Filename:
%Date:          7/15/2008
%Programmer:    Chris Hoops
%
%Description:
%
%User interface for the cylinder pressure post-processing
%Allows user to enter mutiple data points to be processed in a single
data
%group

%**********************************************************************
****

clear
clear all
close all
clc

tic

%****************** Data Processing Parameters to adjust
*****************
%**********************************************************************
****

% group where data is to be processed
group = '2';

%Points to be processed
point = [ 43 ];


%**********************************************************************
****
%**********************************************************************
****

for z = 1:length(point);
    fprintf('\n')
    disp(['                          Group ' num2str(group) ' Point '
num2str(point(z))])
    fprintf('\n')
    rangemin = 5 + 18 * ( point(z) - 1 );
```

```
    rangemax = 20 + 18 * ( point(z) - 1 );
    eval(['[allfile' num2str(point(z)) '] =
xlsread(''D:\GM_Data\Spark_hook_data\updated_Engine_excel_sheet_act'','
'Group ' group ''',''j' num2str(rangemin) ':j' num2str(rangemax)
''');'])

    rangecal = 5 + 18 * ( point(z) - 1 );
    eval(['[calslow' num2str(point(z)) '] =
xlsread(''D:\GM_Data\Spark_hook_data\updated_Engine_excel_sheet_act'','
'Group ' group ''',''p' num2str(rangecal) ':p' num2str(rangecal)
''');'])
    eval(['[num' num2str(point(z)) ',calfast' num2str(point(z)) '] =
xlsread(''D:\GM_Data\Spark_hook_data\updated_Engine_excel_sheet_act'','
'Group ' group ''',''q' num2str(rangecal) ':q' num2str(rangecal)
''');'])

    for k = 1:16;
        if eval(['allfile' num2str(point(z)) '(k);']) == 0

        else
           eval(['files' num2str(point(z)) '(k) = allfile'
num2str(point(z)) '(k);'])
        end
    end

    if eval(['allfile' num2str(point(z)) '(1);']) == 0;

disp('**********************************************************************
******');
        disp(' There are no data files for this point. Check Excel
Spreadsheet!!!');

disp('**********************************************************************
******');
    elseif eval(['calslow' num2str(point(z)) '(1);'])  == 0;

disp('**********************************************************************
******');
        disp('The calibration file for the slow data is missing check
Excel sheet!!!');

disp('**********************************************************************
******');
    elseif eval(['num' num2str(point(z))]) == 0;

disp('**********************************************************************
******');
        disp('The calibration file for the fast data is missing check
Excel sheet!!!');

disp('**********************************************************************
******');
```

```
    else

    for a = 1:eval(['length(files' num2str(point(z)) ');']);

         [data data_cycle] = data_process( ['time_data_raw_'
eval(['num2str(calslow' num2str(point(z)) ')']) '.lvm'] , [
'data_ind_cal_' eval(['num2str(calfast' num2str(point(z)) '{1,1})'])
'.lvm' ] , ['time_data_raw_' eval(['num2str(files' num2str(point(z))
'(a))']) '.lvm'],...
             ['data_ind_' eval(['num2str(files' num2str(point(z))
'(a))']) '.lvm']);


         eval(['grp' group '_pt' num2str(point(z)) '.cad.case'
num2str(a) ' = data.CAD;']);
         eval(['grp' group '_pt' num2str(point(z)) '.avg.case'
num2str(a) ' = data.AVG;']);
         eval(['grp' group '_pt' num2str(point(z)) '.Diag.case'
num2str(a) ' = data.Diagnostics;']);
         eval(['grp' group '_pt' num2str(point(z)) '.header =
data.header;']);
         eval(['grp' group '_pt' num2str(point(z)) '.Units =
data.Units;']);
         eval(['grp' group '_pt' num2str(point(z)) '.Data_warnings =
data.Data_warning;']);

         eval(['grp' group '_pt' num2str(point(z)) '_cycle.case'
num2str(a) ' = data_cycle;']);

         disp([num2str(a/ eval(['length(files' num2str(point(z))
')*.01'])) ' % Complete']);

         fprintf('\n')

         eval(['sum_diag = sum(grp' group '_pt' num2str(point(z))
'.Diag.case' num2str(a) ');']);

         eval(['diag_length = length(grp' group '_pt' num2str(point(z))
'.Diag.case' num2str(a) ');'])

         for c = 1:diag_length,

             eval(['error_code = grp' group '_pt' num2str(point(z))
'.Diag.case' num2str(a) '(' num2str(c) ');'])

             if error_code == 1,

                 eval(['error_displ = grp' group '_pt'
num2str(point(z)) '.header.Diag(' num2str(c) ');'])
                 disp(error_displ)
```

```
            else

            end

            clear error_code error_displ

        end

        clear diag_length

        if sum_diag > .5
           fprintf('\n')

disp('**********************************************************************
******');
            eval(['disp('                    Possible data error for
case ' num2str(a) ''')'])

disp('**********************************************************************
******');
        else
        end

        clear sum_diag data data_cycle

    end

    eval(['grp' group '_pt' num2str(point(z)) '.cases = ' num2str(a)
';']);

    cd('D:\GM_Data\Spark_hook_data\Matlab_files')

    savefile = ['grp' num2str(group) '_pt' num2str(point(z))];
    eval(['save ' savefile '.mat ' savefile ' ' savefile '_cycle'])

    cd('D:\GM_Data\Spark_hook_data')

    end

    eval(['cases = grp' num2str(group) '_pt' num2str(point(z))
'.cases;'])


    for g = 1:cases;

    eval(['MAP_check = grp' num2str(group) '_pt' num2str(point(z))
'.avg.case' num2str(g) '.Ecu(3);'])
    eval(['Intake_check = mean(grp' num2str(group) '_pt'
num2str(point(z)) '.cad.case' num2str(g) '.Int(10:end,1));'])
```

```
    MAP_intake_diff = abs(MAP_check - Intake_check);

       if MAP_intake_diff > 5,


disp('**************************************************************
******');
           eval(['disp(''The difference between the intake pressure and
MAP is large for case ' num2str(g) '!!!'')']);

disp('**************************************************************
******');

       else

       end

    clear cases MAP_check Intake_check MAP_intake_diff

    end

    clear rangemin rangemax rangecal
    eval(['clear allfile' num2str(point(z)) ' calfast'
num2str(point(z)) ' calslow' num2str(point(z)) ' files'
num2str(point(z)) ])
    eval(['clear grp' group '_pt' num2str(point(z)) ' grp' group '_pt'
num2str(point(z)) '_cycle' ])
end

toc
```

Appendix I: Cylinder Pressure Post-Processing Code and Diagnostics

```
function [data data_cycle] = data_process(fn_P_T_cal, fn_P_fast_cal,
fn_test, fn_ind);

%GM Modeling Data Processing
%*********************************************************************
****
%
%Filename:
%Date:          7/22/2008
%Programmer:    Chris Hoops
%
%Description:
%
%fn_P_T_cal - filename for optional pressure offset and optional
temperature calibration
%                   pressure calibration is recommended
%fn_test - filename of time-based test data to be processed
%fn_ind - filename of indicating data associated with fn_test
%*********************************************************************
****

plot_cntrl = 0;                     %Flag to enable (=1) plots

%unit conversions

inHg_2_kPa = 3.386388;
bar_2_kPa = 100;
bar_2_Pa = 100000;
Pa_2_bar = 1 / 100000;
kPa_2_Pa = 1000 ;
kPa_2_bar = 1 / 100;
psi_2_kPa = 6.89475728;
Pa_2_kPa = 1 / 1000;
ft_2_m = 0.3048;
lb_2_N = 4.4482216;
p_cyl_v_2_bar = 50;
l_2_m3 = 0.001;
mm_2_m = 0.001;
deg_2_rad = pi/180;

%Calibration Options

P_cal = 1;      %calibrate all the pressures based on barometric
pressure sensor
T_cal = 0;

% Option to fix intake pressure due to wrong gain in the amplifier
(data up
```

```
% to 8-28-2008 data file 1195)
% 1 is for old data with bad gain
% 0 is for data after file 1195 (has correct gain)

Intake_cal = 1;

Intake_cal_fix = 4.468 / 5.458;     % coversion for wrong gain
Intake_cal_offset = .50 ;       % offset due to gain adjustment


%pressure pinning method for cylinder pressure
% 1 = pinning of cylinder pressure to the average of exhaust and intake
% pressure during valve overlap
% 2 = pinning of cylinder pressure based on intake pressure of TDC of
% exchaust stroke
% 3 = pinning of cylinder pressure to average of exhaust pressure
during
% the exhaust stroke 5 deg after EVO and 5 deg before IVO

Press_pin_method = 3;


%Load data files
data_cal = mean(dlmread(fn_P_T_cal));
data_tst = mean(dlmread(fn_test));
data_cal_ind = mean(dlmread(fn_P_fast_cal));

%Intake and Exhaust Pressure Calibrations

M_INT = 2.5;    %slope, V/bar
B_INT = 2.5;    %offset, V @ atmospheric
M_EXH = 2.5;    %slope, V/bar
B_EXH = 2.5;    %offset, V @ atmospheric

%****************************************
%Adjust Pressures to reference condition
%****************************************

P_engine_offset = 0;
P_post_filter_offset = 0;
if P_cal == 1,

    %Convert master baro sensor into kPa
    P_master = data_cal(67) * inHg_2_kPa;

    %Apply Slopes to pressure sensors and converts to kPa

    P_post_filter_nom = ( 26 + data_cal(44) * (32 - 26) / 4.9981 ) *
inHg_2_kPa;
    P_engine_inlet_nom = data_cal(43) / 5 * psi_2_kPa * 30;
```

```
    P_avg_exh_nom = data_cal(45) / 5 * psi_2_kPa * 30;
    P_oil_nom = (( data_cal(68) * 1.6984 + 1.15994) * 12 ) * psi_2_kPa;

    % Determines intake pressure based on option above due to the
incorrect
    % or correct gain

    if Intake_cal == 1
        P_intake_nom = ( (data_cal_ind(10)* ( Intake_cal_fix ) +
Intake_cal_offset)) / M_INT  * bar_2_kPa;
    else
        P_intake_nom = ( data_cal_ind(10) / M_INT ) * bar_2_kPa;
    end

    P_exhuast_nom = ( data_cal_ind(12) / M_EXH ) * bar_2_kPa;



    %Determine offset correction to calibrate to master

    P_engine_offset = P_master - P_engine_inlet_nom;
    P_post_filter_offset = P_master - P_post_filter_nom;
    P_intake_offset = P_master - P_intake_nom;
    P_exhaust_offset = P_master - P_exhuast_nom;
    P_oil_offset = P_master - P_oil_nom;
    P_avg_exh_offset = P_master - P_avg_exh_nom;


    %determines if there is a large difference between master pressure
    %during calibration to catch sensor errors.

    if (abs(P_engine_offset) > 8 ) | (abs(P_post_filter_offset) > 8 )|
(abs(P_intake_offset) > 8 )| (abs(P_exhaust_offset) > 8
)|abs(P_oil_offset > 8) | abs(P_avg_exh_offset > 8)

disp('**************************************************************
*****');
        disp('Large pressure correction required for baro correction.
Check data!!!');

disp('**************************************************************
******');
    end

    clear P_engine_inlet_nom P_post_filter_nom P_master P_avg_exh_nom
P_oil_nom P_intake_nom P_exhuast_nom
end

%****************************************
%Adjust temperatures to reference condition
%****************************************
```

```
T_offset(1:27) = 0;
if T_cal == 1,

    b = 0;
    T_sum = 0;
    for a = 1:27,

        if data_cal(a+1) > 10 & data_cal(a+1) < 40,
            b = b + 1;
            T_sum = T_sum + data_cal(a+1);
        end
    end

    T_master = T_sum / b;
    T_offset = [ 1  T_master - data_cal(2:27)];

end

%****************************************
%Update offset for LFE Measurement
%****************************************

LFE_V_offset = data_cal(65);

%Process Data

%Temperature Data

T_test_cell = data_tst(2) + T_offset(2);
T_pre_cat = data_tst(3) + T_offset(3);
T_post_cat = data_tst(4) + T_offset(4);
T_oil_sump = data_tst(5) + T_offset(5);
T_cat_brick = data_tst(6) + T_offset(6);
T_exh_run_1 = data_tst(7) + T_offset(7);
T_exh_run_2 = data_tst(8) + T_offset(8);
T_exh_run_3 = data_tst(9) + T_offset(9);
T_exh_run_4 = data_tst(10) + T_offset(10);
T_exh_man = data_tst(11) + T_offset(11);
T_cool_out = data_tst(12) + T_offset(12);
T_high_bay = data_tst(13) + T_offset(13);
T_cool_in = data_tst(14) + T_offset(14);
T_int_run_1 = data_tst(15) + T_offset(15);
T_int_run_2 = data_tst(16) + T_offset(16);
T_int_run_3 = data_tst(17) + T_offset(17);
T_int_run_4 = data_tst(18) + T_offset(18);
T_int_wall_1 = data_tst(19) + T_offset(19);
T_int_wall_2 = data_tst(20) + T_offset(20);
T_int_wall_3 = data_tst(21) + T_offset(21);
T_int_wall_4 = data_tst(22) + T_offset(22);
T_int_man = data_tst(23) + T_offset(23);
```

```
T_inlet = data_tst(24) + T_offset(24);
T_post_filt = data_tst(25) + T_offset(25);
T_LFE = 0.5 * (data_tst(26) + T_offset(26) + data_tst(27) +
T_offset(27));

%Emissions Data

H2 = data_tst(34);
NOX = data_tst(35);
THC = data_tst(36);
CO2 = data_tst(37);
COH = data_tst(38);
COL = data_tst(39);
O2 = data_tst(40);
UEGO_pre = data_tst(63);

%Dyno Data

N_dyno = data_tst(41);
T_dyno = data_tst(42);

%Pressure Data
P_high_bay = data_tst(67) * inHg_2_kPa;                %Pressure of
highbay
P_baro_inlet = ( 26 + data_tst(43) * (32 - 26) / 4.9981 ) ...
    * inHg_2_kPa + P_engine_offset ;
P_post_filter = data_tst(44) *  206.842 / 4.9486 ...
    + P_post_filter_offset;
P_oil = (( data_cal(68) * 1.6984 + 1.15994) * 12 ) * psi_2_kPa +
P_oil_offset;
P_Exh_avg = (data_tst(45) * 30 / 5) * psi_2_kPa + P_avg_exh_offset;


%Humidity

RH_high_bay = data_tst(66);

%Laminar Flow Element

mu = [ 10   1.03034
12.77777778 1.02253
15.55555556 1.01487
18.33333333 1.00736
21.11111111 1
23.88888889 0.99277
26.66666667 0.98568
29.44444444 0.97872
32.22222222 0.97189
35  0.96518
37.77777778 0.9586
40.55555556 0.95213
```

```
43.33333333 0.94578
46.11111111 0.93953
48.88888889 0.9334];


P_LFE = (data_tst(65) - LFE_V_offset) * 10/4.9909;
% MAF_LFE = 3.53544e01 * P_LFE - 1.95440e-1 * P_LFE^2 *
interp1(mu(:,1),mu(:,2), T_LFE)...
%     * 1.293 * P_baro_inlet / 101.325 * 273.15 / (273.15 + T_LFE) *
28.317 / 60;

MAF_LFE = ( 3.53544e01 * P_LFE - 1.95440e-1 * P_LFE^2 *
interp1(mu(:,1),mu(:,2), T_LFE))...
    * 1.293 * P_baro_inlet / 101.325 * 273.15 / (273.15 + T_LFE) *
28.317 / 60;


%ETAS Data

EQR_ECU = data_tst(46);
N_ECU = data_tst(47);
MAP_ECU = data_tst(48);
APC_ECU = data_tst(49);
SPRK_ECU = data_tst(50);
ICAM_ECU = data_tst(51);
ECAM_ECU = data_tst(52);
T_COOL_ECU = data_tst(53);
THROT_PCT_PSTN = data_tst(54);
THROT_PCT_AREA_widle = data_tst(55);
THROT_PCT_AREA_wIdleLrn = data_tst(56);
MAF_ECU = data_tst(57);


%**********************************************************************
****
%Cylinder Pressure Measurment Processing
%**********************************************************************
***(

CYCLE_NUM = 200;                    %Number of cycles to ensemble
average
NUM_CYL = 4;                        %Number of cylinders

pmark = ['k' 'b' 'r' 'y'];                                %Plot markers
leg = ['Cyl. 1' ; 'Cyl. 2' ; 'Cyl. 3' ; 'Cyl. 4'];      %Legend labels

c_P_CYL = [ 2 4 6 8];       %data file columns for cylinder pressure
c_P_INT = 10;               %data file columbn for intake pressure
c_P_EXH = 12;               %data file column for exhaust pressure
```

```
%Encoder Phasing Error Correction - A positive value for ENC_ERR will
delay
%the pressure trace, a negative value will retard the pressure trace.
If
%the encoder pulse is early, then it needs advanced to align it with
TDC.

%Encoder error due mechanical setup and measurement
%TDC pulse early = negative ENC_ERR_CAL term
%TDC pulse late = positive ENC_ERR_CAL term

ENC_ERR_CAL = 0.1;

%Encoder error due to inverted A signal
%TDC pulse early = negative ENC_ERR_signal term
%TDC pulse late = positive ENC_ERR_signal term

ENC_ERR_signal = 0.75;

%Encoder error due to time delay of encoder
%TDC pulse early = negative ENC_ERR_delay term
%TDC pulse late = positive ENC_ERR_delay term
%converts time into degrees based on RPM

delay = 10.7e-6 ;   % Time based delay due to acquisition (s)
deg_per_rev = 360 ;


ENC_ERR_delay = delay * (mean(N_dyno) / 60 ) * deg_per_rev ;

%Total encoder error;
%TDC pulse early = negative ENC_ERR term
%TDC pulse late = positive ENC_ERR term

ENC_ERR = ENC_ERR_CAL + ENC_ERR_signal + ENC_ERR_delay ;

%Load data file

DATA = dlmread(fn_ind, '\t', [0 0 CYCLE_NUM*720-1 13]);


%Select Relevant cylinder Data and Apply Sensor Calibrations and Filter

CAD = 0:719;

[B1 A1] = butter(4,.9);

for i = 1:NUM_CYL,

    data = DATA(:,c_P_CYL(i) ) * p_cyl_v_2_bar;
```

```
    x(:,i) = data;

    P_cyl(:,i) = filtfilt( B1, A1, data) * bar_2_Pa;

end

clear x


% For the intake and exhuast pressures 2.5 V corresponds to atmospheric
% pressure so the offset is not needed when converting V to bar


%loads intake pressure and applies correct gain (units in bar)

if Intake_cal == 1;
    P_int = ((( DATA(:,c_P_INT) + Intake_cal_offset) * (
Intake_cal_fix) / M_INT ) + ( P_intake_offset * kPa_2_bar ));
else
    P_int = (( DATA(:,c_P_INT)  / M_INT ) + ( P_intake_offset *
kPa_2_bar ));
end

P_exh = (( DATA(:,c_P_EXH)  / M_EXH ) + ( P_exhaust_offset * kPa_2_bar
));

CRNK_CNT = DATA(:,14);

clear data;
clear DATA;


%Calculate Volume Vector from Engine Geometry

theta = CAD * deg_2_rad;

r_c = 10.29994856;      %compression ratio
V_d = 0.596057176;      %cylinder displacement volume (L)
V_c = V_d / (r_c - 1);  %clearance volume (L)
l = 143.8;              %connecting rod length (mm)
a = 49;                 %crank radius (mm)
B = 88;                 %Bore (mm)
L = 98;                 %stroke (mm)
R = l/a;                %ratio of connecting rod length to crank radius

V_d = V_d * l_2_m3;
V_c = V_c * l_2_m3;
l = l * mm_2_m;
a = a * mm_2_m;
B = B * mm_2_m;
```

```
L = L * mm_2_m;

%Volume Calculation, Heywood pg 44

V = V_c * (1 + 0.5*(r_c - 1)*(R + 1 - cos(theta - pi) - (R^2 -
(sin(theta - pi)).^2).^.5));
V = circshift(V',0);
V = V';

%This for loop picks out the location of the maximum pressure for each
%cycle in the data set.  This data is useful to determine if the
encoder is
%either false triggering or not triggering on every degree.  The first
is
%indicated by the peak pressure location increasing with increasing
cycles.
%The latter would indicated by the opposite.  The loop normalizes each
%cycle and shifts the cycle to have TDC fire at 360 degrees.  This
corrects
%any errros that would arrise from pressures being 360 deg out of
phase.
%Once the maximum pressure is found the max pressure CAD location is
shifted back 360
%degrees if it was shifted due to out of phase condition.

%shift used to get all cylinders referenced to 360 fire to find max
press
%location

TDC_shift_max =  [ 2 1 -1 0] * 180;


for i = 1:NUM_CYL,

    for a = 1:CYCLE_NUM,

        P_cycle = P_cyl( ((a-1)*720+1):(a*720), i );
        P_cycle = circshift(P_cycle, TDC_shift_max(i));
        P_cycle_max = max(P_cycle);
        P_cycle_min = min(P_cycle);
        P_cycle = ( P_cycle - P_cycle_min ) ./ ( P_cycle_max -
P_cycle_min);

        cad_location = find(P_cycle == 1);

        if cad_location >= 540 | cad_location <= 180;

            P_cycle = circshift(P_cycle,360);
            [cycle_peak(a,i), cad_peak(a,i)] = max(P_cycle);
            cad_peak(a,i) = cad_peak(a,i) - 360;
```

```
        else

            [cycle_peak(a,i), cad_peak(a,i)] = max(P_cycle);

        end


        cycle(a) = a;

    end

    cad_peak(:,i) = cad_peak(:,i) - TDC_shift_max(i);


      if plot_cntrl == 1,
          figure(1);
          plot(cycle, cad_peak(:,i), [ pmark(i) '.'], cycle,
polyval(peak_poly, cycle), pmark(i));
          hold on;
      end
end

%detects if the DAQ system falls behind in recording and discards
corrupted
%data


for i = 1:NUM_CYL,

    CAD_peak_avg(i) = mean(cad_peak(1:50,i));

    for a = 1:CYCLE_NUM,

        DAQ_lag = abs(cad_peak(a,i) - CAD_peak_avg(i));

        if DAQ_lag > 50

            CYCLE_lag(i) = a;
            break

        else
            CYCLE_lag(i) = 200 ;
        end

    end

end
```

```
if min(CYCLE_lag) < 200

    CYCLE_NUM = min(CYCLE_lag) - 2;

else
end


clear CYCLE_lag DAQ_lag CAD_peak_avg
cycle = cycle(1:CYCLE_NUM);

for i = 1:NUM_CYL,

        peak_poly = polyfit(cycle', cad_peak(1:CYCLE_NUM,i), 1);
        peak_slope(i) = peak_poly(1,1);

end

% averages peak pressure for all 200 cycles of cylinder one

cyl_1_CAD = mean(cad_peak(:,1));

%if the mean of the max pressure is negative to to 360 degree shift
back
%then it moves max pressure to the end of the cycle.

if cyl_1_CAD < 0

    cyl_1_CAD = 720 + cyl_1_CAD;

end

if plot_cntrl == 1,

    figure(1);
    title('Location of Peak Pressure for Each Cycle');
    ylabel('Peak Pressure Location (CAD)');
    xlabel('Cycle Number');
    grid on;

end

clear peak_poly P_cycle;

%*****************************************************************
****
%The following loop performs a number of operations to ensemble average
the
%data, translate it to a common CAD reference, reference the cylinder
%pressure
```

```
%*********************************************************************
****

%Parameters to shift all cylinders to same CAD reference

TDC_shift =  [ 1 0 -2 -1] * 180;

%Rotates data 360 in case it is out of phase due to z-index problem


if cyl_1_CAD > 250 & cyl_1_CAD < 500,
    TDC_shift = TDC_shift + 360;
end


%Interpolate between points to compensate for encoder alignment error

CAD_full = 0:143999;

yo = [0:1:719];
figure
y = P_cyl(1:720,1);
plot(yo,y);grid on
title('Cylinder Pressure for 1 Cycle Before and After Encoder Shift')
xlabel('CAD [deg]')
ylabel('Pressure [bar]')
hold on

for i = 1:NUM_CYL,

    P_cyl(:,i) = interp1(CAD_full, P_cyl(:,i), CAD_full - ENC_ERR,
'linear');

end

z = P_cyl(1:720,1);
plot(yo,z,'r')
legend('Cyl Press. w/ Encoder Err','Cyl Press. w/ Corrected Enc Err')


P_int(:,1) = interp1(CAD_full, P_int(:,1), CAD_full - ENC_ERR,
'linear');
P_exh(:,1) = interp1(CAD_full, P_exh(:,1), CAD_full - ENC_ERR,
'linear');


%Drop out first and last cycle of data due to filter distortion

P_cyl = P_cyl(720:length(P_cyl)-720,:);
P_int = P_int(720:length(P_int)-720,:);
```

```
P_exh = P_exh(720:length(P_exh)-720,:);

%decreases cycle number by two since two cycles are removed above

CYCLE_NUM = CYCLE_NUM - 2 ;

%CAD bounds for intake pressure pinning of cylinder pressure
%This is for the LE5 engine and compensates for cam motion
%to determine the valve overlap period

TDC_low = 541 - 2 * ICAM_ECU ;  %IVO
TDC_high = 543 + 2 * ECAM_ECU ;  %EVC

%determines EVO based on cam duration and EVC.

cam_duration = 240 ; % degrees
EVO_pinning = TDC_high - cam_duration ;


for a = 1:NUM_CYL,

    %Generate Shifted Vector for all signals with TDC fire at 180
degrees

    P_cyl_s(:,a) = circshift(P_cyl(:,a), TDC_shift(a));
    P_int_s(:,a) = circshift(P_int, TDC_shift(a));
    P_exh_s(:,a) = circshift(P_exh, TDC_shift(a));


    %Determine mean pressure of cylinders and reference pressure
    %for each cycle in the data for each cylinder "a"
    %over each cycle "b"


    % method 1 from above...pinning of cylinder pressure to the average
of exhaust and intake
    % pressure during valve overlap

    if Press_pin_method == 1;

        for b = 1:CYCLE_NUM,

            low = ceil( (b-1) * 720 + TDC_low );
            high = floor( (b-1) * 720 + TDC_high );

            P_mean_OL(b,a) = mean( .5 * P_int_s(low:high,a) + 0.5 *
P_exh_s(low:high,a));
            P_cyl_OL(b,a) = mean(P_cyl_s(low:high,a));
```

```
        end


       %Calculate the pressure offset for pinning each cycle of data

        P_cyl_offset = ( P_mean_OL * bar_2_Pa) - P_cyl_OL ;       %Pa

        clear low high P_mean_OL P_cyl_OL

    % method 2 from above...pinning of cylinder pressure based on
intake pressure of TDC of
    % exchaust stroke

    elseif Press_pin_method == 2;

        for b = 1:CYCLE_NUM,

            P_mean_int_TDC(b,a) = mean(P_int_s((b-1)*720+538:(b-
1)*720+542));
            P_mean_cyl_TDC(b,a) = mean(P_cyl_s((b-1)*720+540:(b-
1)*720+540));

        end

       %Calculate the pressure offset for pinning each cycle of data

        P_cyl_offset = ( P_mean_int_TDC * bar_2_Pa) - P_mean_cyl_TDC;
%Pa
        clear P_mean_int_TDC P_mean_cyl_TDC

    % method 3 from above...pinning of cylinder pressure to average of
exhaust pressure during
    % the exhaust stroke 5 deg after EVO and 5 deg before IVO


    elseif Press_pin_method == 3;

        for b = 1:CYCLE_NUM,

            low = ceil( (b-1) * 720 + ( 440 ) );
            high = floor( (b-1) * 720 + ( 480 ) );

            P_mean_exh_overlap(b,a) = mean(P_exh_s(low:high,a));
            P_mean_cyl_overlap(b,a) = mean(P_cyl_s(low:high,a));

        end

        P_cyl_offset = ( P_mean_exh_overlap * bar_2_Pa ) -
P_mean_cyl_overlap ;
```

```
        clear low high
    end

end

clear P_mean_exh_overlap P_mean_cyl_overlap


%Apply pressure offset over each cycle

for a = 1:NUM_CYL,
    for b = 1:CYCLE_NUM,

    P_cyl_cycle(((b-1)*720 + 1):(b*720),a) = P_cyl_s(  ((b-1)*720 +
1):(b*720) ,a) + P_cyl_offset(b,a);

    end
end


for a = 1:NUM_CYL,
    P_cyl_cycle(:,a) = circshift(P_cyl_cycle(:,a), -1*TDC_shift(a));
end


%Calculate Ensemble Average of Data

for i = 1:NUM_CYL,

    for a = 1:720;

        P_cyl_avg(a,i) = 0;
        P_int_avg(a,i) = 0;
        P_exh_avg(a,i) = 0;

        if i == 1,
            CRNK_CNT_avg(a) = 0;
        end

        for b = 1:CYCLE_NUM;

            P_cyl_avg(a,i) = P_cyl_avg(a,i) + 1/CYCLE_NUM * P_cyl_s(
(b-1)* 720 + a, i) + P_cyl_offset(b,i) ./ (CYCLE_NUM)  ;
            P_int_avg(a,i) = P_int_avg(a,i) + 1/CYCLE_NUM * P_int_s(
(b-1)* 720 + a, i);
            P_exh_avg(a,i) = P_exh_avg(a,i) + 1/CYCLE_NUM * P_exh_s(
(b-1)* 720 + a, i);
```

```
        if i ==1,
            CRNK_CNT_avg(a) = CRNK_CNT_avg(a) + 1/CYCLE_NUM * CRNK_CNT(
(b-1)* 720 + a, i);
        end

        end
    end

end


P_cyl_avg(:,5) = 0.25 * ( P_cyl_avg(:,1) + P_cyl_avg(:,2) +
P_cyl_avg(:,3) + P_cyl_avg(:,4) );
P_exh_avg(:,5) = 0.25 * ( P_exh_avg(:,1) + P_exh_avg(:,2) +
P_exh_avg(:,3) + P_exh_avg(:,4) );
P_int_avg(:,5) = 0.25 * ( P_int_avg(:,1) + P_int_avg(:,2) +
P_int_avg(:,3) + P_int_avg(:,4) );

%Repair first point because interpolation gives bad value
P_cyl_avg(1,:) = 0.5 * P_cyl_avg(720,:) + 0.5 * P_cyl_avg(2,:);


%Diagnostics for CAD slip or additional points added

for i = 1:NUM_CYL,

    if abs(peak_slope(i)) > .1

        peak_slope_diag(i) = 1;

    else

        peak_slope_diag(i) = 0;

    end

end

% determines max pressure for each cylinder and difference between max
and
% min

for a = 1:NUM_CYL,

    max_CYL_press(a) = max(P_cyl_avg(:,a)) ;

end

CYL_press_range = max( max_CYL_press) - min( max_CYL_press) ;
```

```
if CYL_press_range > 1000000;

    CYL_press_range_diag = 1 ;

else

    CYL_press_range_diag = 0 ;

end

%Pressure limits for diagnostics

for a = 1:NUM_CYL;

    if max(P_cyl_avg(:,a)) > ( 10000 * kPa_2_Pa ) | max(P_cyl_avg(:,a))
< (400 * kPa_2_Pa);
        P_cyl_max_diag(a) = 1;
    else
        P_cyl_max_diag(a) = 0;
    end

    if  min(P_cyl_avg(:,a)) < 0;
        P_cyl_min_diag(a) = 1;
    else
        P_cyl_min_diag(a) = 0;
    end

end



    if mean(P_exh_avg(:,1)) > 1.8 | mean(P_exh_avg(:,1)) < .95;
        P_exh_diag = 1;
    else
        P_exh_diag = 0;
    end

    if mean(P_int_avg(:,1)) > 1 | mean(P_int_avg(:,1)) < .12;
        P_int_diag = 1;
    else
        P_int_diag = 0;
    end


%Calculate IMEP values for pumping and combustion stroke

for i = 1:NUM_CYL,

    imep_c(i) = (trapz(V(1:360),P_cyl_avg(1:360,i)) ./ V_d) * Pa_2_kPa
;
```

```
    imep_p(i) = (trapz(V(361:720),P_cyl_avg(361:720,i)) ./ V_d) *
Pa_2_kPa;
    imep(i) = imep_c(i) + imep_p(i);


    % Checks limits for IMEP

    if (imep(i) > 1700) | (imep(i) < 0);
        imep_diag(i) = 1;
    else
        imep_diag(i) = 0;
    end

end

% finds the IMEP values for each cycle of the engine

for i = 1:NUM_CYL;

    for b = 1:CYCLE_NUM - 4;

        imep_c_cycle(b,i) = (trapz(V(1:360),P_cyl_cycle(((b-1)*720 +
1):((b*720)-360),i)) ./ V_d) * Pa_2_kPa ;
        imep_p_cycle(b,i) = (trapz(V(361:720),P_cyl_cycle(((b-1)*720 +
361):(b*720),i)) ./ V_d) * Pa_2_kPa ;
        imep_tot_cycle = imep_c_cycle + imep_p_cycle ;

    end

end

%determines the coverience in the IMEP values on a cycle by cycle basis


for i = 1:NUM_CYL;

%     imep_c_cycle_cov(i) = (std(imep_c_cycle(:,i)) / imep_c(i)) * 100;
%     imep_p_cycle_cov(i) = (std(imep_p_cycle(:,i)) / imep_p(i)) * 100;
    imep_tot_cycle_cov(i) = (std(imep_tot_cycle(:,i)) / imep(i)) * 100;

    if imep_tot_cycle_cov(i) > 10;

        imep_cov_diag(i) = 1;
    else
        imep_cov_diag(i) = 0;
    end


end
```

```
% calculates BMEP value

BMEP = ((4*pi*T_dyno)/(4*V_d)) * ft_2_m * lb_2_N * Pa_2_kPa;

%Compares the BMEP to IMEP for diagnostics

for a = 1:NUM_CYL;

    IMEP_BMEP_diff = abs(imep(a) - BMEP);

    if IMEP_BMEP_diff > 300 ;
        IMEP_BMEP = 1 ;
    else
        IMEP_BMEP = 0 ;
    end
end

%creates error code if num of cycles used for averages is too low

if CYCLE_NUM < 50

    CYCLE_diag = 1 ;

else
    CYCLE_diag = 0 ;
end

%**********************************************************************
****
%The following loop plots out some basic figures with the pressure data
%**********************************************************************
****


if plot_cntrl == 1,

        figure;
        plot(CAD, P_cyl_avg);
        hold on;
        plot([180 180],[0 max(P_cyl_avg(:,i))*1.25], 'k-');
        plot([360 360],[0 max(P_cyl_avg(:,i))*1.25], 'k-');
        plot([540 540],[0 max(P_cyl_avg(:,i))*1.25], 'k-');
        legend([ leg ]);
        ylabel('Cyl. Pres. (bar)');
        grid on;
        axis([0 720 0 round( max(max(P_cyl_avg) / 10 )) * 10 ]);

end
```

```
%Converts Intake and exhuast pressures to pascals

P_int = P_int * bar_2_Pa;
P_exh = P_exh * bar_2_Pa;


data.Diagnostics = [ imep_diag IMEP_BMEP P_cyl_max_diag P_cyl_min_diag
P_exh_diag P_int_diag CYL_press_range_diag peak_slope_diag CYCLE_diag
];
data.header.Diag = {'IMEP Cyl 1 limit' 'IMEP Cyl 2 limit' 'IMEP Cyl 3
limit' 'IMEP Cyl 4 limit' 'IMEP - BMEP diff.'...
    'Cyl 1 max Press limit' 'Cyl 2 max Press limit' 'Cyl 3 max Press
limit' 'Cyl 4 max Press limit' 'Cyl 1 min Press limit'...
    'Cyl 2 min Press limit' 'Cyl 3 min Press limit' 'Cyl 4 min Press
limit' 'exh press limit' 'intake press limit' 'Max Cylinder Pressure
Variation' 'CAD max Press. Slope Cyl 1' 'CAD max Press. Slope Cyl 2'
'CAD max Press. Slope Cyl 3' 'CAD max Press. Slope Cyl 4'...
    'Cycles used for CAD averages'};


data.AVG.Pres = [P_high_bay P_baro_inlet P_post_filter P_oil
P_Exh_avg];

data.header.Pres = {'P_high_bay' 'P_baro_inlet' 'P_post_filter' 'P_oil'
'P_Exh_avg'};

data.Units.Pres = {'kPa' 'kPa' 'kPa' 'kPa' 'kPa'};

data.AVG.Temp = [  T_LFE ...
    T_cat_brick T_cool_in T_cool_out T_dyno T_exh_man T_exh_run_1
T_exh_run_2 T_exh_run_3...
    T_exh_run_4 T_high_bay T_inlet T_int_man T_int_run_1 T_int_run_2
T_int_run_3 T_int_run_4...
    T_int_wall_1 T_int_wall_2 T_int_wall_3 T_int_wall_4 T_oil_sump
T_post_cat...
    T_post_filt T_pre_cat T_test_cell];

data.Units.Temp = {'deg C' 'deg C' 'deg C' 'deg C' 'deg C'...
    'deg C' 'deg C' 'deg C' 'deg C'...
    'deg C' 'deg C' 'deg C' 'deg C' 'deg C' 'deg C' 'deg C' 'deg C'...
    'deg C' 'deg C' 'deg C' 'deg C' 'deg C' 'deg C'...
    'deg C' 'deg C' 'deg C'};

data.header.Temp = {'T_LFE' 'T_cat_brick' 'T_cool_in' 'T_cool_out'
'T_dyno'...
    'T_exh_man' 'T_exh_run_1' 'T_exh_run_2' 'T_exh_run_3'...
    'T_exh_run_4' 'T_high_bay' 'T_inlet' 'T_int_man' 'T_int_run_1'
'T_int_run_2' 'T_int_run_3' 'T_int_run_4'...
    'T_int_wall_1' 'T_int_wall_2' 'T_int_wall_3' 'T_int_wall_4'
'T_oil_sump' 'T_post_cat'...
    'T_post_filt' 'T_pre_cat' 'T_test_cell'};
```

```
data.AVG.Ecu = [EQR_ECU N_ECU MAP_ECU APC_ECU SPRK_ECU ICAM_ECU
ECAM_ECU T_COOL_ECU THROT_PCT_PSTN THROT_PCT_AREA_widle...
    THROT_PCT_AREA_wIdleLrn MAF_ECU];

data.AVG.IMEP.Pump_avg = [imep_p];
% data.AVG.IMEP.Pump_cov = [imep_p_cycle_cov];
data.AVG.IMEP.Comb_avg = [imep_c];
% data.AVG.IMEP.Comb_cov = [imep_c_cycle_cov];
data.AVG.IMEP.Total_avg = [imep];
data.AVG.IMEP.Total_cycle = [imep_tot_cycle];
data.AVG.IMEP.Total_cov = [imep_tot_cycle_cov];
data.Units.IMEP.Pump_avg = {'kPa' 'kPa' 'kPa' 'kPa'};
data.Units.IMEP.Comb_avg = {'kPa' 'kPa' 'kPa' 'kPa'};
data.Units.IMEP.Total_avg = {'kPa' 'kPa' 'kPa' 'kPa'};
data.Units.IMEP.Total_cycle = {'kPa' 'kPa' 'kPa' 'kPa'};
data.Units.IMEP.Total_cov = {'%' '%' '%' '%'};
data.header.IMEP.Pump_avg = {'Pumping IMEP Cyl 1 from Ensembled
average' 'Pumping IMEP Cyl 2 from Ensembled average' 'Pumping IMEP Cyl
3 from Ensembled average' 'Pumping IMEP Cyl 4 from Ensembled average'};
data.header.IMEP.Comb_avg = {'Combustion IMEP Cyl 1 from Ensembled
average' 'Combustion IMEP Cyl 2 from Ensembled average' 'Combustion
IMEP Cyl 3 from Ensembled average' 'Combustion IMEP Cyl 4 from
Ensembled average'};
data.header.IMEP.Total_avg = {'Total IMEP Cyl 1 from Ensembled average'
'Total IMEP Cyl 2 from Ensembled average' 'Total IMEP Cyl 3 from
Ensembled average' 'Total IMEP Cyl 4 from Ensembled average'};
data.header.IMEP.Total_cycle = {'Total IMEP Cyl 1 cycle by cycle'
'Total IMEP Cyl 2 cycle by cycle' 'Total IMEP Cyl 3 cycle by cycle'
'Total IMEP Cyl 4 cycle by cycle'};
data.header.IMEP.Total_cov = {'Cycle by cycle covariance Cyl 1' 'Cycle
by cycle covariance Cyl 2' 'Cycle by cycle covariance Cyl 3' 'Cycle by
cycle covariance Cyl 4'};


data.AVG.BMEP = [BMEP];
data.header.BMEP = {'Brake Mean Effective Pressure'};
data.Units.BMEP = {'kPa'};


data.header.Ecu = {'EQR_ECU' 'N_ECU' 'MAP_ECU' 'APC_ECU' 'SPRK_ECU'
'ICAM_ECU'...
    'ECAM_ECU' 'T_COOL_ECU' 'THROT_PCT_PSTN' 'THROT_PCT_AREA_widle'...
    'THROT_PCT_AREA_wIdleLrn' 'MAF_ECU'};

data.Units.Ecu = {'Lambda' 'RPM' 'kPa' 'mg' 'Deg BTDC (crank angle)'
'Cam Deg'...
    'Cam Deg' 'deg C' 'Percent' 'Percent'...
    'Percent' 'g/s'};

data.AVG.Emis = [H2 NOX THC CO2 COH COL O2 UEGO_pre];
```

```
data.header.Emis = {'H2' 'NOX' 'THC' 'CO2' 'COH' 'COL' 'O2'
'UEGO_pre'};

data.Units.Emis = {'Volume %' 'ppm' 'ppm' 'volume %' 'volume %' 'ppm'
'volume %' 'phi'};

data.AVG.Dyno = [N_dyno T_dyno];

data.header.Dyno = {'N_dyno' 'T_dyno'};

data.Units.Dyno = {'RPM' 'ft-lb'};

data.AVG.Misc = [RH_high_bay P_LFE MAF_LFE];

data.header.Misc = {'RH_high_bay' 'P_LFE' 'MAF_LFE'};

data.Units.Misc = {'Percent Humidity' 'in H2O' 'g/s'};

data.CAD.Int = P_int_avg * bar_2_kPa;
data.CAD.Exh = P_exh_avg * bar_2_kPa;
data.CAD.Cyl = P_cyl_avg * Pa_2_kPa;
data.CAD.CAD_slope =  peak_slope ;
data.CAD.Max_CYL_Press = max_CYL_press * Pa_2_kPa;
data.CAD.CYCLE_NUM = CYCLE_NUM;


data.header.CYCLE_NUM = { 'Engine Cycles used for CAD averages' };
data.header.Int = { 'P_int_1' , 'P_int_2' , 'P_int_3', 'P_int_4',
'P_int_avg'};
data.header.Exh = { 'P_exh_1' , 'P_exh_2' , 'P_exh_3', 'P_exh_4',
'P_exh_avg'};
data.header.Cyl = { 'P_cyl_1' , 'P_cyl_2' , 'P_cyl_3', 'P_cyl_4',
'P_cyl_avg'};
data.header.CAD_slope = { 'Slope CAD max Press. Cyl 1' 'Slope CAD max
Press. Cyl 2' 'Slope CAD max Press. Cyl 3' 'Slope CAD max Press. Cyl 4'
};
data.header.Max_CYL_Press = {'Max Pressure Cyl 1' 'Max Pressure Cyl 2'
'Max Pressure Cyl 3' 'Max Pressure Cyl 4' };

data.Units.Cyl = { 'kPa' , 'kPa' , 'kPa', 'kPa', 'kPa'};
data.Units.Exh = { 'kPa' , 'kPa' , 'kPa', 'kPa', 'kPa'};
data.Units.Int = { 'kPa' , 'kPa' , 'kPa', 'kPa', 'kPa'};
data.Units.CAD_slope = { 'deg/sample' 'deg/sample' 'deg/sample'
'deg/sample' };
data.Units.Max_CYL_Press = { 'kPa' 'kPa' 'kPa' 'kPa' };

data.Data_warning = {'No Known Problems'};

data_cycle.Cyl = P_cyl_cycle * Pa_2_kPa;
```

```
data_cycle.Int = P_int * Pa_2_kPa;
data_cycle.Exh = P_exh * Pa_2_kPa;
data_cycle.Vol = V;
```

Appendix J: Inverse Thermodynamic Model User Interface

```
%
%  Script for burn rate estimation from experimental data.
%
%  Chris Hoops, September 2008
%
%  NOTE: the script operates on a specified engine operating condition,
and
%  processes the burn rate for all the cases (spark hooks) contained.
%  The Burn Rate Function is saved in a data structure similar to the
one
%  used to process the LE5 data.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
clear all
close all
clc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Part 1: Select engine operating condition (data points)

[filename, pathname] = uigetfile('*.mat', 'Pick an M-
file','MultiSelect', 'on');

if iscell(filename) == 0

    [Cyl,inputs,outputs,timing] = retrieve(filename,pathname);

    RGF = zeros(size(inputs,1),1);
    inputs = cat(2,inputs,RGF);
    clear RGF

    disp(sprintf('Number of Data Points: %d',size(Cyl,1)));
    disp('Calculating burn rate: press enter to continue.....');
    pause
    clc
    close all

    % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Loads corresponding .mat file into the workspace to save burn
    % rate data to the structure.
```

```
    load(strcat(pathname,filename));

    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Run inverse thermodynamic model to determine burn ratio for each
of
    % the cases.

    for j=1:size(Cyl,1)

    %Create string array of case names
    if j<10
        case2=(['00' num2str(j)]);
    elseif j>=10 && j<100
        case2=(['0' num2str(j)]);
    elseif j>=100
        case2=(['' num2str(j)]);
    end

    disp(sprintf('Processing case %d...',j));
    [data,MEP(j,:),Pmax(j,:),SOI(j,:),comb_CA(j,:),eff(j)] = inv_TD
(squeeze(Cyl(j,:,:)),inputs(j,:),outputs(j,:),timing(j,:),1,1);
    eval([ filename(1:end-4) '.Burn_Rate.BR.Case' case2 '(:,1) =
data(:,1);']);
    eval([ filename(1:end-4) '.Burn_Rate.BR.Case' case2 '(:,2) =
data(:,2);']);
    eval([ filename(1:end-4) '.Burn_Rate.BR.Case' case2 '(:,3) =
data(:,3);']);
    eval([ filename(1:end-4) '.Burn_Rate.MEP.Case' case2 '= MEP('
num2str(j) ',:);']);
    eval([ filename(1:end-4) '.Burn_Rate.Pmax.Case' case2 '= Pmax('
num2str(j) ',:);']);
    eval([ filename(1:end-4) '.Burn_Rate.SOI.Case' case2 '= SOI('
num2str(j) ',:);']);
    eval([ filename(1:end-4) '.Burn_Rate.Comb_CA.Case' case2 '=
comb_CA(' num2str(j) ',:);']);
    eval([ filename(1:end-4) '.Burn_Rate.Eff.Case' case2 '= eff('
num2str(j) ');']);


    end


    % Labels burn rate header and units

    eval([ filename(1:end-4) '.Burn_Rate.BR.header = {''Deg from spark
timing to EVO [Deg]'' ''Heat Release Rate [1/CAD]'' ''Cumlative Heat
Release [normalized]''};' ])
    eval([ filename(1:end-4) '.Burn_Rate.MEP.header = {''Gross IMEP IVC
- EVO [bar]'' ''Total IMEP full cycle [bar]'' ''BMEP [bar]''};' ])
```

```
    eval([ filename(1:end-4) '.Burn_Rate.Pmax.header = {''Max press
[bar]'' ''CAD of max Press [deg]'' ''Max rise rate [bar/deg]'' ''CAD of
rise rate [deg]''} ;' ])
    eval([ filename(1:end-4) '.Burn_Rate.SOI.header = {''CAD of 1%
burned (from spark timing) [CAD]'' ''Cyl Pres. at 1% [bar]'' ''temp. at
1% [deg C]''};' ])
    eval([ filename(1:end-4) '.Burn_Rate.comb_CA.header = {''5% burn
[CAD]'' ''10% burn [CAD]'' ''15% burn [CAD]'' ''25% burn [CAD]'' ''50%
burn [CAD]'' ''75% burn [CAD]'' ''90% burn [CAD]''} ;' ])
    eval([ filename(1:end-4) '.Burn_Rate.eff.header = {''Combustion
efficiency [%]''};' ])

    % Saves burn rate to corresponding structures
    cd(pathname);
    eval(['save ' filename ' ' filename(1:end-4)])

    % Clears files in workspace
    eval(['clear ' filename(1:end-4) ])
    disp('...Done')
    close all

else

    for i=1:length(filename),

        [Cyl,inputs,outputs,timing] = retrieve(filename{i},pathname);

        RGF = zeros(size(inputs,1),1);
        inputs = cat(2,inputs,RGF);
        clear RGF

        disp(sprintf('Number of Data Points: %d',size(Cyl,1)));
        disp('Calculating burn rate: press enter to continue.....');
        pause(.1)
        close all



        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Loads corresponding .mat file into the workspace to save burn
        % rate data to the structure.

        temp = filename{i};
        load(strcat(pathname,temp));

        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % Run inverse thermodynamic model to determine burn ratio for
each of
        % the cases.
```

```
        for j=1:size(Cyl,1)

            %Create string array of case names
            if j<10
                case2=(['00' num2str(j)]);
            elseif j>=10 && j<100
                case2=(['0' num2str(j)]);
            elseif j>=100
                case2=(['' num2str(j)]);
            end

            [data,MEP(j,:),Pmax(j,:),SOI(j,:),comb_CA(j,:),eff(j)] =
inv_TD (squeeze(Cyl(j,:,:)),inputs(j,:),outputs(j,:),timing(j,:),1,1);
            eval([ filename(1:end-4) '.Burn_Rate.BR.Case' case2 '(:,1)
= data(:,1);']);
            eval([ filename(1:end-4) '.Burn_Rate.BR.Case' case2 '(:,2)
= data(:,2);']);
            eval([ filename(1:end-4) '.Burn_Rate.BR.Case' case2 '(:,3)
= data(:,3);']);
            eval([ filename(1:end-4) '.Burn_Rate.MEP.Case' case2 '=
MEP(' num2str(j) ',:);']);
            eval([ filename(1:end-4) '.Burn_Rate.Pmax.Case' case2 '=
Pmax(' num2str(j) ',:);']);
            eval([ filename(1:end-4) '.Burn_Rate.SOI.Case' case2 '=
SOI(' num2str(j) ',:);']);
            eval([ filename(1:end-4) '.Burn_Rate.Comb_CA.Case' case2 '=
comb_CA(' num2str(j) ',:);']);
            eval([ filename(1:end-4) '.Burn_Rate.Eff.Case' case2 '=
eff(' num2str(j) ');']);
            pause(.1)

        end

        % Labels burn rate header and units

        eval([ filename(1:end-4) '.Burn_Rate.BR.header = {''Deg from
spark timing to EVO [Deg]'' ''Heat Release Rate [1/CAD]'' ''Cumlative
Heat Release [normalized]''};' ])
        eval([ filename(1:end-4) '.Burn_Rate.MEP.header = {''Gross IMEP
IVC - EVO [bar]'' ''Total IMEP full cycle [bar]'' ''BMEP [bar]''};' ])
        eval([ filename(1:end-4) '.Burn_Rate.Pmax.header = {''Max press
[bar]'' ''CAD of max Press [deg]'' ''Max rise rate [bar/deg]'' ''CAD of
rise rate [deg]''} ;' ])
        eval([ filename(1:end-4) '.Burn_Rate.SOI.header = {''CAD of 1%
burned (from spark timing) [CAD]'' ''Cyl Pres. at 1% [bar]'' ''temp. at
1% [deg C]''};' ])
        eval([ filename(1:end-4) '.Burn_Rate.comb_CA.header = {''5%
burn [CAD]'' ''10% burn [CAD]'' ''15% burn [CAD]'' ''25% burn [CAD]''
''50% burn [CAD]'' ''75% burn [CAD]'' ''90% burn [CAD]''} ;' ])
        eval([ filename(1:end-4) '.Burn_Rate.eff.header = {''Combustion
efficiency [%]''};' ])
```

```
        % Saves burn rate to corresponding structures
        path1 = cd;
        cd(pathname);
        eval(['save ' temp ' ' temp(1:end-4) ' ' temp(1:end-4)
'_cycle'])
        cd(path1);

        % Clears files in workspace
        eval(['clear ' temp(1:end-4) ])
        clear data MEP Pmax SOI comb_CA eff temp inputs outputs timing
Cyl
        disp('...Done')
        close all
    end
    clc
    clear filename pathname
end


%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
```

Retrieve Function

```
function [ Cyl, inputs, outputs, timing ] = retrieve(filename,pathname)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Function to extract selected data from engine operating points. For
any
% given condition, the function extracts data for all cases (spark
points)
% contained in the corresponding structure.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

load(strcat(pathname,filename));
eval(strcat('caseno = length(',filename(1:end-4),'.GoodCases(:,1));'));


% Generate high-speed data structure: each array contains cylinder-
averaged
% pressure trace from -180 to 540 (so that 0CAD -> TDC firing),
together
% with cylinder volume, derivative and cylinder area.
Phase = -180:539 ;                    % Defines vector of CAD
```

```
[Vcyl,dVcyl,Acyl]=volume_LE5(Phase);    % Calculates volume and
derivative

% Plotting utility, to verify correctness of cylinder pressure data and
% valve phasing
figure('color',[1 1 1],'position',[5 260 770 605]);

%Number of Cylinders used for shifting
NUM_CYL = 4;

%Shift scheme used for each cylinder to get TDC fire at 180 deg
deg_shift = [ 0 -180 360 180];

for i=1:caseno;

    %Create string array of case names
    if i<10
        case2=(['00' num2str(i)]);
    elseif i>=10 && i<100
        case2=(['0' num2str(i)]);
    elseif i>=100
        case2=(['' num2str(i)]);
    end



    eval(['Cyl_ind(:,1) = (' filename(1:end-4),'.Cylinder1.Case_'
(case2) '(:,2));'])  %bar
    eval(['Cyl_ind(:,2) = (' filename(1:end-4),'.Cylinder2.Case_'
(case2) '(:,2));'])  %bar
    eval(['Cyl_ind(:,3) = (' filename(1:end-4),'.Cylinder3.Case_'
(case2) '(:,2));'])  %bar
    eval(['Cyl_ind(:,4) = (' filename(1:end-4),'.Cylinder4.Case_'
(case2) '(:,2));'])  %bar

    %Shifts the cylinder pressures to have TDC fire at 180 deg
    for n=1:NUM_CYL;
        Cyl_shift(:,n) = circshift(Cyl_ind(:,n),deg_shift(n));
    end

    %finds the average of the four cylinder pressures
    for j=1:length(Cyl_shift(:,1));
        Cyl(i,j,2) = mean(Cyl_shift(j,:));
    end

    eval(['MAF(i) = ',filename(1:end-4),'.RLT.Case_' (case2) '(3);'])
% kg/s
    eval(['MAP(i) = mean(',filename(1:end-4),'.IM.Case_' (case2)
'(:,2));'])  % bar
    eval(['RPM(i) = ',filename(1:end-4),'.ControlVars.Case_' (case2)
'(1);'])  % RPM
```

```
    eval(['TPS(i) = ',filename(1:end-4),'.ControlVars.Case_' (case2)
'(2);'])  % Percent
    eval(['ICAM(i) = ',filename(1:end-4),'.ControlVars.Case_' (case2)
'(3);']) % ECU cam angle
    eval(['ECAM(i) = ',filename(1:end-4), '.ControlVars.Case_' (case2)
'(4);']) % ECU cam angle
    eval(['Spark(i) = ',filename(1:end-4),'.ControlVars.Case_' (case2)
'(5);']) % deg BTDC
    eval(['EQR(i) = ',filename(1:end-4),'.ECU.Case_' (case2) '(1);']) %
Equivalence Ratio
    eval(['Torque(i) = ',filename(1:end-4),'.RLT.Case_' (case2)
'(2);'])  % Nm
    eval(['BMEP(i) = ',filename(1:end-4),'.RLT.Case_' (case2) '(10);'])
% bar
    % Use IMEP calculated for four cylinders and average
    eval(['IMEP(i) = ',filename(1:end-4),'.RLT.Case_' (case2) '(9);'])
% kPa to bar
    AFR(i) = ( 1 / EQR(i) ) * 14.7 ;
    Cyl(i,:,1) = Phase;
    Cyl(i,:,3) = Vcyl ;
    Cyl(i,:,4) = dVcyl ;
    Cyl(i,:,5) = Acyl ;

    %Calculates the average intake runner temperature and coolant
    %temperature

    eval(['T_int_run(i) = (',filename(1:end-4),'.RLT.Case_' (case2)
'(16)) - 273.15;'])  % K to C
    eval(['T_cool(i) = ',filename(1:end-4),'.RLT.Case_' (case2)
'(9);'])  % C

    % Calculate [EVO EVC IVO IVC] based on cam profile and timing.
Values are
    % phased so that 0CAD => TDC Firing
    th_0 = [49 193 174 326]; % EVO EVC IVO IVC, based on GT-Power
notation (parked cam timing)
    timing(i,:) = round([(th_0(1)+ECAM(i))*2 (th_0(2)+ECAM(i))*2
(th_0(3)-ICAM(i))*2 (th_0(4)-ICAM(i))*2-720]);

    subplot(211)
    plot(Phase,Cyl(i,:,2),'g');
    hold on;
    grid on;

    H=line([-Spark(i),-Spark(i)],[0 5]);
    set(H,'color','m','linewidth',2);

    subplot(212)
    plot(1e6*Cyl(i,:,3),Cyl(i,:,2),'g');
    hold on;
    grid on;
```

```
    disp(sprintf('Case No.: %d, IMEP = %g bar, BMEP = %g
bar;',i,IMEP(i),BMEP(i)));

    clear Cyl_ind
end
subplot(211)

H=line([timing(1,1) timing(1,1)],[0 5]);
set(H,'color','r','linewidth',2);
text(timing(1,1)+5,5,'EVO');
H=line([timing(1,2) timing(1,2)],[0 5]);
set(H,'color','r','linewidth',2);
text(timing(1,2)+5,5,'EVC');
H=line([timing(1,3) timing(1,3)],[0 5]);
set(H,'color','b','linewidth',2);
text(timing(1,3)+5,5,'IVO');
H=line([timing(1,4) timing(1,4)],[0 5]);
set(H,'color','b','linewidth',2);
text(timing(1,4)+5,5,'IVC');

text(-Spark(end)+5,5,'Spark');
xlabel('Crank Angle [deg]');
ylabel('Cylinder Pressure [bar]');
title(['Group ',filename(6:7),sprintf(': RPM=%d, TPS=%d, ICAM=%d,
ECAM=%d',round(RPM(1)),...
round(TPS(1)),round(ICAM(1)),round(ECAM(1)))]);
subplot(212)
xlabel('Volume [cm^3]');
ylabel('Cylinder Pressure [bar]');

inputs = [ RPM' AFR' TPS' -ICAM' ECAM' -round(Spark') T_int_run'
T_cool' ];
outputs = [ MAF' MAP' IMEP' Torque' BMEP' ];

eval(['clear ',filename(1:end-4)]);
```

Appendix K: Inverse Thermodynamic Model and Functions

```
function [data,MEP,Pmax,SOI,comb_CA,eff] = inv_TD
(Cyl,inputs,outputs,timing,flag_gamma,flag_ht)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%  Marcello Canova, February 2008, updated September 2008
%
%  This function estimates the gross and net heat release rate from
%  pressure trace analysis
%
```

```
%  NEW APRIL 08: Included residual gas fraction in properties
calculation
%  NEW SEPTEMBER 08: Reviewed I/O structure, added option of
calculating
%                    burn rate using ideal gas (constant gamma) and
without
%                    heat transfer
%
% *Inputs
%  Cyl          =  Data Structure: [Theta,Pcyl,Vcyl,dVcyl,Acyl] (720
points)
%  inputs       =  [RPM,AFR,TPS,ICAM,ECAM,Spark,residuals]
%  outputs      =  [MAF MAP IMEP Torque BMEP]
%  timing       =  [EVO EVC IVO IVC]
%  flag_gamma   =  '1' for caloric equation, '0' sets gamma=1.35
%  flag_ht      =  '1' sets Woschni model, '0' for no heat transfer
%
%
%* Outputs
%  data         = [Theta(IVC:EVO) Pressure FBR FB]
%  h_sp         = spark timing index (from IVC)
%  eff          = approximated combustion efficiency
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

% Parameters definition
T_im    = 40+273;  %inputs(7)        % Estimated intake manifold
temperature [K]
Tcool   = 85+273;  %inputs(8)        % Estimated coolant temperature
[K]
Hvi     = 44.1*10^6;       % Lower heating value of the fuel [J/kg]

% Process input data
N      = inputs(1);        % Engine speed [r/min]
M_air  = outputs(1);       % Air mass flow rate [kg/s]
AFR    = inputs(2);        % Air/fuel ratio
M_fuel = M_air/AFR;        % Fuel mass flow rate [kg/s]
M_tot  = M_air + M_fuel;   % Total mass flow rate [kg/s]
X_res  = inputs(9);        % Residual gas fraction (used to calculate
properties)

Mf     = 120*M_fuel/(4*N); % Mass of fuel per cylinder per cycle [kg]
Mcyl   = 120*M_tot/(4*N);  % Total mass per cylinder per cycle [kg]

% Process in-cylinder data
Theta  = Cyl(:,1);    % Crank Angle [deg]
P_cyl  = Cyl(:,2)*1e5;% Cyl. Pressure [Pa]
V_cyl  = Cyl(:,3);    % Cyl. Volume [m^3]
A_cyl  = Cyl(:,5);    % Cyl. Area [m^2]

% Determines array indexes for IVC, EVO and Spark Timing
```

```
h_IVC = find(Theta==timing(4));
h_EVO = find(Theta==timing(1));
Spark = inputs(6);                % Spark Timing, positive if bTDC
h_sp = find(Theta==Spark);        % array index corresponding to spark
timing

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% In-Cylinder Temperature Calculation (uses caloric equation)
[Cp,Cv]=ftherm3(T_im,X_res); % First approximation for Cp, Cv
R=Cp-Cv;
k=Cp./Cv;
T=(P_cyl.*V_cyl)/(Mcyl*R);
switch flag_gamma
    case 1
    [Cp,Cv]=ftherm3(T,X_res);    % Second calculation of Cp,Cv
(vectors)
    case 0
    Cp = 1019*ones(size(T));
    Cv = 758*ones(size(T));
end

Cp

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Heat Release Analysis: isolate array portions from IVC to EVO
Th=Theta(h_IVC:h_EVO);
P=P_cyl(h_IVC:h_EVO);
V=V_cyl(h_IVC:h_EVO);
T=T(h_IVC:h_EVO);
Cp=Cp(h_IVC:h_EVO);
Cv=Cv(h_IVC:h_EVO);
A=A_cyl(h_IVC:h_EVO);
h_sp = h_sp-h_IVC+1;                    % Spark timing relative to IVC

% Calculation of net and gross HRR from energy equation
dL=P.*gradient(V,Th);                   % Net Displacement Work
dU=Mcyl*Cv.*gradient(T,Th);             % Internal Energy Variation

switch flag_ht
    case 1
        hw = convection([P,T,V],N);
    case 0
        hw = zeros(size(T));
end
hcool=2000;
Tw=(hw.*T+Tcool*hcool)./(hw+hcool);
Qw=hw.*A.*(T-Tw);           % Convection heat rate [kW]
dQw=Qw/(6*N);

dQn=dU+dL;                  % Net Heat Release Rate [J/deg]
```

```
[yf] = fourier(Th,dQn,inputs(1),1000);
dQn = yf;

dQg=dQn+dQw;                % Gross Heat Release Rate [J/deg]
FBR=dQg/(Mf*Hvi);           % Burn Rate Function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Plot Results
% figure('color',[1 1 1],'position',[5 260 770 605]);


% Isolate portion of P, T, HRR and FBR from spark timing to EVO
Th_new = 0:1:length(Th(h_sp:end))-1;
P = P(h_sp:end);
T = T(h_sp:end);
dQn = dQn(h_sp:end);
dQg = dQg(h_sp:end);
FBR = FBR(h_sp:end);


% Normalize and correct burn rate function (removes negative parts,
adjusts
% to monotonic function and limits value between 0 and 1)

FB=cumtrapz(FBR);
[i,j]=min(cumtrapz(FBR));
[h,k]=max(cumtrapz(FBR));
FB=(cumtrapz(FBR)-i)/(h-i);      % Normalize to 1
FB(1:j)=0;                       % Correct negative values
FB(k:end)=1;                     % Forces function to 1 at combustion
end
FB=sort(FB);                     % Imposes monotonic behavior



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Data Processing: calculates metrics for evaluating combustion
process:
%     IMEPg, IMEPn and comparison with BMEP
%     Peak Pressure and related CAD
%     Max. Pressure Gradient and related CAD
%     Start of Ignition (0.1% of burn rate), with related P and T
%     CA5, CA10, CA15, CA25, CA50, CA75, CA90
%     Combustion efficiency

IMEPg=trapz(dL)/(2.4e-3/4)*1e-5;                          % Gross IMEP
(Accuracy Check)
```

```
IMEPn=trapz(P_cyl.*gradient(V_cyl,Theta))/(2.4e-3/4)*1e-5; % Net IMEP
(Accuracy Check)
MEP = [IMEPg, IMEPn, outputs(end)];
disp(sprintf('Check on MEP:  IMEPg = %g bar,  IMEPn = %g bar,  BMEP =
%g bar.',IMEPg,IMEPn,outputs(end)));

[y,i] = max(1e-5*P);
Pmax = [y Th_new(i)];
[y,i] = max(gradient(1e-5*P,Th_new));
Pmax = [Pmax y Th_new(i)];
disp(sprintf('Check on Pressure:  max(P) =%g bar @ %d CAD,  max(dP/dx)
= %g bar/deg @ %g CAD.',Pmax(1),Pmax(2),Pmax(3),Pmax(4)));

i = find(FB >=0.01);
i=i(1);
SOI = [Th_new(i),1e-5*P(i),T(i)-273];
disp(sprintf('Check on Ignition Timing: Spark = %dCAD, SOI = %d CAD,  P
= %g bar, T= %g ^oC.',Spark,SOI(1),SOI(2),SOI(3)));

CAxx = [5 10 15 25 50 75 90]/100;
for i=1:length(CAxx)
    a=find(FB>=CAxx(i));
    b=a(1)-1;
    comb_CA(i)=Th_new(b);
end
disp(sprintf('Check on Burn Angle:  CA10 = %ddeg, CA50 = %ddeg, CA75 =
%ddeg.',comb_CA(2),comb_CA(5),comb_CA(6)));

eff = trapz(FBR);                  % Estimate of comb. Efficiency (Not
100% accurate, use just as feedback!!)
disp(sprintf('Check on Combustion Efficiency:  %g',eff));
disp('-----------------------------------------------------------------
----');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Create output array with crank angle, pressure, FBR and BR defined
from
% spark timing (h_sp) to exhaust valve opening(h_EVO)

data=[Th_new',FBR,FB];
```

Specific Heat Function

```
%*******************************************************
%  FUNCTION FOR THERMODYNAMIC CONSTANTS CALCULATION
%*******************************************************
%   References: Heywood: "Internal Combustion Engine Fundamentals",
```

```
%               CRC Handbook of Chemistry and Physics
%
function [Cp,Cv,h]=ftherm3(T,inert)
%*****************************************************************
****
%*Input Variables:
%  T     = mixture temperature                                  [K]
%  inert = inert gas fraction                                   [%]
%
%*Output Variables:
%  h     = specific enthalpy of the mixture                     [J/kg]
%  Cp    = specific heat at constant pressure of the mixture    [J/kg K]
%  Cv    = specific heat at constant volume of the mixture      [J/kg K]
%*****************************************************************
****

% Fifth order Polynomial
k1=[6.8806e-016 -8.5303e-014
   -3.6573e-012  5.7064e-010
    7.586e-009  -1.4483e-006
   -7.8089e-006  0.0016526
    0.0044958       -0.61388
   -0.55271      1089.2];
k2=[6.8806e-016 -8.5303e-014
   -3.6573e-012  5.7064e-010
    7.586e-009  -1.4483e-006
   -7.8089e-006  0.0016526
    0.0044958       -0.61388
   -0.5178       800.7];
k3=[3.2338e-013 4.5664e-011
   -1.5012e-009  -2.3678e-007
    2.6069e-006 0.00044464
   -0.0017809      -0.26723
    0.64568     1079.3
   -20294          -6.3418e+005];

inert=[inert;1];
Cp=zeros(size(T));
Cv=zeros(size(T));
h=zeros(size(T));
for i=1:size(k1,1)
    Cp=Cp+k1(i,:)*inert.*T.^(size(k1,1)-i);
    Cv=Cv+k2(i,:)*inert.*T.^(size(k2,1)-i);
    h=h+k3(i,:)*inert.*T.^(size(k3,1)-i);
end
```

Heat Release Model Function

```
function [hw] = convection(data,N)
```

```
% Convection coefficient based on traditional Woschni model (Heywood)
% *Input:
%  data=[pressure [Pa]; temperature [K]; Volume [m^3]]
%  N: Engine speed [r/min]
%
% *Output:
%  hw: convection coefficient [W/m^2K]
%
%*********************************************************************
****


P=data(:,1);
T=data(:,2);
V=data(:,3);

% LE5 geometric parameters (change according to engine specifications)
B=0.088;          %Cylinder bore [m]
L=0.098;          % Stroke [m]

Vd=0.25*pi*B^2*L; % Displaced Volume [m^3]
Sp=2*L*N/60;      % Mean piston speed [m/s]
k=1.35;           % Specific gas constant

% Woschni Constants (enter values from Heywood or GT-Power)
C1=2.28;
C2=3.24e-3;
C=3.26;
m=0.8;

% Calculation of the term proportional to mean cylinder gas velocity:
Vref=V(1);
pref=P(1);
Tref=T(1);
p_mot=pref*((Vref./V).^k); % Motored pressure
w = C1*Sp + C2*(Vd*Tref/(Vref*pref))*(P-p_mot);

% Calculation of Heat Transfer Coefficient (Woschni Formula) [W/m^2K]
hw = C*B^(m-1) * (P/1000).^m .* w.^m .* T.^(0.75-1.62*m);
```

Appendix L: Single and Double Wiebe Fitting

```
%
%  Script for to determine burn rate fit
%
%  Chris Hoops, October 2008
%
%  NOTE: the script operates on a specified engine operating condition,
and
```

```
%  processes the burn rate for all the cases (spark hooks) contained.
%  The Burn Rate Function is saved in a data structure similar to the
one
%  used to process the LE5 data.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
clear all
close all
clc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
% Part 1: Select engine operating condition (data points)

[filename, pathname] = uigetfile('*.mat', 'Pick an M-
file','MultiSelect', 'on');

if iscell(filename) == 0,

        filename_num = filename;
        load(strcat(pathname,filename));
        eval(['z = fieldnames(' strcat(filename_num(1:end-
4),'.Burn_Rate.BR') ')']);
        caseno = length(z) - 1;
        clear z x
        eval(['clear ' filename_num(1:end-4) ])


        %Single Wiebe function fitting parameters

        for n = 1:caseno,

            [Theta, BR, comb_CA] = retrieve_burn(filename,pathname,n);

            %intial guess for optimizing functions
            x0 = [5,2,4,5,2];

            %options associated with optimization
            options = optimset('TolFun',1e-4,'TolX',1e-
4,'MaxIter',1e4,'MaxFunEvals',1e10);

            %bounds for the optimization
            LB = [ 0 0 0 .01 .01];
            UB = [ 1 200 200 100 100];

            %optimizing function

            [x2,Resnorm(1,1),Residual,Exitflag(1,1)] =
lsqcurvefit(@(in,Theta) Wiebe_2(in, Theta),x0,Theta,BR,LB,UB,options);
```

```
            %Define GT Power Coefficients

            BM=.5;
            BS=.1;
            BE=.9;
            BMC=-log(1-BM);
            BSC=-log(1-BS);
            BEC=-log(1-BE);

            %Define Wiebe Function Parameters
            alpha=x2(1);
            D1=x2(2);
            D2=x2(3);
            E1=x2(4);
            E2=x2(5);

            WC1=(D1/(BEC^(1/(E1+1))-BSC^(1/(E1+1))))^(-(E1+1));
            WC2=(D2/(BEC^(1/(E2+1))-BSC^(1/(E2+1))))^(-(E2+1));
            SOC = 0;

            BR_fit2=alpha.*(1-exp(-(WC1).*(Theta-SOC).^(E1+1)))+(1-
alpha).*(1-exp(-(WC2).*(Theta-SOC).^(E2+1)));

            %intial guess for optimizing functions
            x0 = [5,2];

            %options associated with optimization
            options = optimset('TolFun',1e-4,'TolX',1e-
4,'MaxIter',1e4,'MaxFunEvals',1e4);

            %bounds for the optimization
            LB = [ 0 0 ];
            UB = [ 50 50 ];

            %optimizing function

            [x1,Resnorm(1,1),Residual,Exitflag(1,1)] =
lsqcurvefit(@(in,Theta) single_wiebe(in,
Theta),x0,Theta,BR,LB,UB,options);

            %Define Wiebe Function Parameters
            D1=x1(1);
            E1=x1(2);
            WC1=(D1/(BEC^(1/(E1+1))-BSC^(1/(E1+1))))^(-(E1+1));

            BR_fit1=(1-exp(-(WC1).*(Theta-SOC).^(E1+1)));

            figure
            plot(Theta,BR,'k',Theta,BR_fit1,'--r',Theta,BR_fit2,'--
','linewidth',1.5);grid on
```

```
            title('Single Wiebe Fitted Burn Rate')
            xlabel('CAD [from spark timing]')
            ylabel('Mass Fraction Burned')
            legend('Exp','single','double')

            clear theta_in BR_in
        end

else

    for i = 1:length(filename),

            filename_num = filename{i};
            load(strcat(pathname,filename{i}));
            eval(['z = fieldnames(' strcat(filename_num(1:end-
4),'.Burn_Rate.BR') ')']);
            caseno = length(z) - 1;
            clear z x
            eval(['clear ' filename_num(1:end-4) ])


            %Single Wiebe function fitting parameters

            for n = 1:caseno,

                [Theta, BR, comb_CA] =
retrieve_burn(filename{i},pathname,n);

                %intial guess for optimizing functions
                x0 = [5,2];

                %options associated with optimization
                options = optimset('TolFun',1e-4,'TolX',1e-
4,'MaxIter',1e4,'MaxFunEvals',1e4);

                %bounds for the optimization
                LB = [ 0 0 ];
                UB = [ 50 50 ];

                %optimizing function

                [x,Resnorm(1,1),Residual,Exitflag(1,1)] =
lsqcurvefit(@(in,Theta) single_wiebe(in,
Theta),x0,Theta,BR,LB,UB,options);

                %Define GT Power Coefficients
                BM=.5;
                BS=.1;
                BE=.9;
                BMC=-log(1-BM);
```

```
                BSC=-log(1-BS);
                BEC=-log(1-BE);

                %Define Wiebe Function Parameters
                D1=x(1);
                E1=x(2);
                WC1=(D1/(BEC^(1/(E1+1))-BSC^(1/(E1+1))))^(-(E1+1));
                SOC = 0;

                BR_fit=(1-exp(-(WC1).*(Theta-SOC).^(E1+1)));

                figure
                plot(Theta,BR,Theta,BR_fit,'--r','linewidth',1.5);grid
on
                title('Single Wiebe Fitted Burn Rate')
                xlabel('CAD [from spark timing]')
                ylabel('Mass Fraction Burned')
                legend('Exp','Fitted')

                clear theta_in BR_in
        end

    end

end
```

Retrieve Function

```
function [theta_new, BR, comb_CA] = retrieve_burn(filename, pathname,n)

load(strcat(pathname,filename));

if n<10
    case2=(['00' num2str(n)]);
elseif n>=10 && n<100
    case2=(['0' num2str(n)]);
elseif n>=100
    case2=(['' num2str(n)]);
end

case2

eval(['theta_new = ' filename(1:end-4) '.Burn_Rate.BR.Case'
num2str(case2) '(:,1);' ])
eval(['BR = ' filename(1:end-4) '.Burn_Rate.BR.Case' num2str(case2)
'(:,3);' ])
eval(['comb_CA = ' filename(1:end-4) '.Burn_Rate.Comb_CA.Case'
num2str(case2) ';' ])
```

Single Wiebe Function:

```
function [xb_out] = single_wiebe(in, theta)


%Define GT Power Coefficients
SOC=0;
BM=.5;
BS=.1;
BE=.9;
BMC=-log(1-BM);
BSC=-log(1-BS);
BEC=-log(1-BE);

%Define Wiebe Function Parameters
D1=in(1);
E1=in(2);
WC1=(D1/(BEC^(1/(E1+1))-BSC^(1/(E1+1))))^(-(E1+1));

xb_out=(1-exp(-(WC1).*(theta-SOC).^(E1+1)));
```

Double Wiebe Function

```
function [xb_out]=Wiebe_2(in,Theta)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%  Marcello Canova, April 2008
%
%  Double-Wiebe function for burn rate data fitting
%
%
%* Inputs
%  Params       =  Wiebe Function Parameters (GT-Power Notation)
%  Theta        =  Crank Angle
%  SOC          =  Start of Combustion (Spark Timing)
%
%* Output
%  xb_out       =  Predicted Burn Rate Profile
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

%Define GT Power Coefficients
SOC = 0;
```

```
BM=.5;
BS=.1;
BE=.9;
BMC=-log(1-BM);
BSC=-log(1-BS);
BEC=-log(1-BE);

%Define Wiebe Function Parameters
alpha=in(1);
D1=in(2);
D2=in(3);
E1=in(4);
E2=in(5);

WC1=(D1/(BEC^(1/(E1+1))-BSC^(1/(E1+1))))^(-(E1+1));
WC2=(D2/(BEC^(1/(E2+1))-BSC^(1/(E2+1))))^(-(E2+1));

xb_out=alpha.*(1-exp(-(WC1).*(Theta-SOC).^(E1+1)))+(1-alpha).*(1-exp(-
(WC2).*(Theta-SOC).^(E2+1)));
```